

NAME: Jaime Lima - jdl679@nyu.edu

ML 6143 Prof Sundeep Ragan

PROJECT: NIST CBRS RADAR 3.5 GHz Waveforms Detection

RadarWaveformsML - Detecting the Presence of the Waveforms Immersed in Gaussian Noise using Principal Component Analysis (PCA) and Logistic Regression

Manipulation of Radar and Communication Waveforms using new ML algorithms but not forgetting the Old Tricks

This repository was originally created for a project for the Class "Introduction to Machine Learning at NYU Tandon" ministered by Professor Sundeep Ragan (more details at <https://github.com/sdrangan/introml> (<https://github.com/sdrangan/introml>))

The main goal is to use synthetic radar waveforms (**RF Dataset of Incumbent Radar Systems in the 3.5 GHz CBRS Band**) provided by the scientists at the US National Institute of Standards and Technology (NIST) (more details at <https://data.nist.gov/od/id/mds2-2116> (<https://data.nist.gov/od/id/mds2-2116>)) to investigate the detection of features in the waveforms using *Machine Learning* techniques.

We are interested in simple trainable algorithms that are able to detect the waveforms in the presence of noise (and also fading in the future). We are also interested in find the *signatures* of the waveforms to be able to classify the different radar waveforms and differentiate them from other electromagnetic signals, like the ones used for microwave wireless communications.

Real-time time identification of waveforms is an important tool that can be used to allow coexistence of different emitters sharing the same geographical space. It is also relevant for cyber-security, helping to assure security in cyber-physical environments.

NOTE: to develop the code here we used pieces of code provided in the examples provided in the NIST directory about how read the data files and label files fields correctly in the Python Programming Language. The ML code however is all new based in the class work we did and the class examples provided. Finally, the MATLAB code used by the scientists at NIST to generated radar wavforms similiar to the ones they made available can be found at <https://github.com/usnistgov/SimulatedRadarWaveformGenerator.git/trunk> (<https://github.com/usnistgov/SimulatedRadarWaveformGenerator.git/trunk>)

NIST URLs and Contact Info: <https://data.nist.gov/od/id/mds2-2116> (<https://data.nist.gov/od/id/mds2-2116>)
Contact: Raied Caromi.. Raied Caromi Email: raied.caromi@nist.gov

Importing some Libraries

```
In [1]: import numpy as np
import h5py
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt
GDRIVE_MOUNTED=False
```

Some Details about How to Download RF Dataset of Incumbent Radar Systems in the 3.5 GHz CBRS Band

Each file has 200 synthetically created waveforms, each being 80 ms long. Some waveforms have only gaussian noise. Others have a radar waveform immersered in gaussian noise under several different SNR. Radar waveforms can be either different types of simple pulse trains or more complex chirped waveforms.

Each file size is 2.5 GB. The total dataset is 500 GB! We will use just one file from the database due to memory usage constraints. Additional small csv metadata files provide different types of labels related to the data.

The data can be download directly from NIST using the link [\(https://data.nist.gov/od/id/mds2-2116\)](https://data.nist.gov/od/id/mds2-2116)

For convencience, a subset of of the data was downloaded in the Google drive and is accessible for people from NYU. The shared link below provides access to the NYU folks. The shared folder has the same structure as the NIST database but only few of the large waveform files.

[\(https://drive.google.com/drive/folders/1lhVG7mAMoTFjsXbArvrz3cEIXDIJVnDS?usp=sharing\)](https://drive.google.com/drive/folders/1lhVG7mAMoTFjsXbArvrz3cEIXDIJVnDS?usp=sharing)

Once the code is copied into the person google drive, the code below can be easily adjusted to access the data by chaning the variables *data_folder* and *remote_folder_gdrive*. The code below also allows to run the ipython notebook locally in the machine. In this case the variable *remote_folder_local* needs to be updated

The code below also checks if GPU is enabled. This notebooks can run without GPU.

```
In [2]: str_GPU='GPU'
#str_GPU='CPU' #To fake having a GPU
data_folder="SimulatedRadarWaveforms"
remote_folder_gdrive=".~/gdrive/My Drive/_BIG1/NIST/CBRS/" + data_folder
remote_folder_local="/G/_BIG1/NIST/CBRS/" + data_folder

import os
#!df -k | egrep "Filesystem/overlay"
#!ls

# MOUNT GDRIVE
if 'google.colab' in str(get_ipython()):
    GCOLAB_ENV=True
    print('***RUNNING on CoLab')
    if False==GDRIVE_MOUNTED:
        from google.colab import drive
        drive.mount("/content/gdrive")
        GDRIVE_MOUNTED=True
else:
    print('***NOT RUNNING on CoLab')
    GCOLAB_ENV=False

# CHECK if System has GPU
from tensorflow.python.client import device_lib
resp = device_lib.list_local_devices()
if str_GPU in str(resp):
    has_gpu=True
    gpu_info = !nvidia-smi
    gpu_info = '\n'.join(gpu_info)
    if gpu_info.find('failed') >= 0: print('GPU Info NOT AVAILABLE (!?!)')
    else: print(gpu_info)
else:
    has_gpu=False

# LOAD Data
if True==GCOLAB_ENV:
    data_folder_full = remote_folder_gdrive
    #!rsync -av "$remote_folder" .
    #!du      "$data_folder"
    #!rsync -av "$data_folder" .
else:
    data_folder_full = remote_folder_local

print('Data Folder:', data_folder_full)
!ls -l      "$data_folder_full"

!rm -rf ./flickr2
if (True==has_gpu):
    print("*** DO SOMETHING (HAS GPU)")
else:
    print("*** DO SOMETHING (HAS NO GPU)")
```

```
***NOT RUNNING on CoLab
Data Folder: /G/_BIG1/NIST/CBRS/SimulatedRadarWaveforms
total 3789
-rwxrwxrwx 1 root root 3864830 Nov 19 08:05 allWaveformsTableCombined.csv
-rwxrwxrwx 1 root root      64 Nov 19 08:05 allWaveformsTableCombined.csv.sha
256
-rwxrwxrwx 1 root root      136 Dec  9 21:40 desktop.ini
dr-xr-xr-x 1 root root    4096 Dec  9 21:40 Group1
dr-xr-xr-x 1 root root      0 Dec  9 21:40 Group2
dr-xr-xr-x 1 root root      0 Dec  9 21:40 Group3
dr-xr-xr-x 1 root root      0 Dec  9 21:40 Group4
-rwxrwxrwx 1 root root    2282 Nov 19 08:04 License.txt
-rwxrwxrwx 1 root root      64 Nov 19 08:04 License.txt.sha256
-rwxrwxrwx 1 root root    1399 Nov 19 08:05 ReadMe.txt
-rwxrwxrwx 1 root root      64 Nov 19 08:04 ReadMe.txt.sha256
dr-xr-xr-x 1 root root    4096 Dec  9 21:40 readWaveformsCodeExamples
*** DO SOMETHING (HAS NO GPU)
```

Reading the Radar Waveform Data Set .mat files and the .csv file with the Additional Labels

This will take some time because the files are large.

```
In [3]: #dataRootFolder=data_folder
group_No =1 # 1 to 4
subset_No=2 # 1 to 50
# Preps
dataRootFolder =data_folder_full
fileName      ='group'+str(group_No)+'_subset_'+str(subset_No)+'.mat'
filePath      =Path(dataRootFolder)/('Group'+str(group_No))/(fileName)
waveform_var  ='group'+str(group_No)+'_waveformSubset_'+str(subset_No)
status_var    ='group'+str(group_No)+'_radarStatusSubset_'+str(subset_No)
table_var     ='group'+str(group_No)+'_waveformTableSubset_'+str(subset_No)
infoFileNamePath=Path(dataRootFolder)/('Group'+str(group_No))/('group'+
                           str(group_No)+'_subset_CSVInfo')/(table_var+'.csv')
#
# Read data from .mat file
h5pyObj        = h5py.File(filePath, 'r')
subsetSignals   = h5pyObj[waveform_var][()].view(np.complex)
subsetRadarStatus = h5pyObj[status_var ][()]
#
# Now use Panda to read the info csv file
subsetInfo = pd.read_csv(infoFileNamePath)

print('***DONE')
```

***DONE

Now Printing Some Fields to Get Familiar with the Data

Two label columns of special interest are:

1. h5pyObj[status_varList]: Indicates if a radar waveform is present
2. subsetInfo['BinNo']: Indicates the type of the waveform
3. subsetInfo['SNR']: Indicates the Signal-to-Noise Ratio when the radar waveform is present.

```
In [4]: def P2R(radial, angles):
    return radial * np.exp(1j*angles)

def R2P(x):
    return np.abs(x), np.angle(x)
```

```
In [5]: jj = 1.j
print("fileName      =", fileName)
print("waveform_var=", waveform_var)
print("status_var   =", status_var)
print("table_var    =", table_var)
print("h5pyObj       =", h5pyObj)
print('---')
print('[',waveform_var,',     ] subsetSignals.shape=',subsetSignals.shape)
print('[',status_var ,',     ] subsetRadarStatus.shape=',subsetRadarStatus.shape)
print('subsetInfo.shape=' ,subsetInfo.shape)
print('---')
print('[',waveform_var,',     ] subsetSignals.shape=',subsetSignals.shape)
NPT=5
r, ang = R2P(subsetSignals[0, 0:NPT])
#print(subsetSignals[sigIndex, 0:NPT])
#print('r=\n', r.T)
#print('ang=\n', ang.T)
print(r*np.exp(jj*ang))

fileName      = group1_subset_2.mat
waveform_var= group1_waveformSubset_2
status_var   = group1_radarStatusSubset_2
table_var    = group1_waveformTableSubset_2
h5pyObj      = <HDF5 file "group1_subset_2.mat" (mode r)>
---
[ group1_waveformSubset_2      ] subsetSignals.shape= (200, 800000)
[ group1_radarStatusSubset_2 ] subsetRadarStatus.shape= (200, 1)
subsetInfo.shape= (200, 16)
---
[ group1_waveformSubset_2      ] subsetSignals.shape= (200, 800000)
[1.66089269e-07-2.31654912e-07j 1.35645134e-07+2.35493766e-07j
 1.58416011e-07-1.24614912e-07j 1.21105281e-07+4.92919982e-08j
 4.46692447e-07-3.11198016e-07j]
```

```
In [6]: %% show the info and status of the first 10
#print(subsetRadarStatus[0:20])
print("LABEL RADAR Waveform Present:\n\n", subsetRadarStatus[0:20].ravel())
print("\nOTHER LABELS below:\n")
print("...LABEL NAMES (Columns):\n... ...", subsetInfo.columns)
print("\n... ... LABELS:\n",subsetInfo[0:20])
aux=subsetInfo['SNR']; print("SNRs Values in the file:", aux.unique());
aux=subsetInfo['BinNo'];print("WAVEFORMS in the file:", aux.unique());
print('***DONE QUICKLY')
```

LABEL RADAR Waveform Present:

```
[0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0]
```

OTHER LABELS below:

```
...LABEL NAMES (Columns):
... ... Index(['BinNo', 'PulseWidth', 'PulsesPerSecond', 'PulsesPerBurst',
    'ChirpWidth', 'ChirpDirection', 'SamplingFrequency', 'ActualPulseWidth',
    'PhaseCodingType', 'SUID', 'radarStatus', 'radarSignalCenterFreq',
    'radarSignalStartTime', 'SNR', 'NoisePowerdBmPerMHz', 'duration'],
    dtype='object')
```

... ... LABELS:

	BinNo	PulseWidth	PulsesPerSecond	PulsesPerBurst	ChirpWidth	\
0	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	
2	Q3N#3	0.000085	800.0	20.0	50000000.0	
3	P0N#2	0.000026	1690.0	10.0	NaN	
4	NaN	NaN	NaN	NaN	NaN	
5	NaN	NaN	NaN	NaN	NaN	
6	NaN	NaN	NaN	NaN	NaN	
7	NaN	NaN	NaN	NaN	NaN	
8	NaN	NaN	NaN	NaN	NaN	
9	NaN	NaN	NaN	NaN	NaN	
10	NaN	NaN	NaN	NaN	NaN	
11	NaN	NaN	NaN	NaN	NaN	
12	Q3N#1	0.000003	1620.0	20.0	50000000.0	
13	NaN	NaN	NaN	NaN	NaN	
14	Q3N#1	0.000004	870.0	22.0	70000000.0	
15	P0N#1	0.000002	1010.0	35.0	NaN	
16	NaN	NaN	NaN	NaN	NaN	
17	Q3N#2	0.000021	1750.0	2.0	3000000.0	
18	NaN	NaN	NaN	NaN	NaN	
19	NaN	NaN	NaN	NaN	NaN	

	ChirpDirection	SamplingFrequency	ActualPulseWidth	PhaseCodingType	\
0	NaN	10000000	NaN	NaN	
1	NaN	10000000	NaN	NaN	
2	Down	10000000	0.000085	NaN	
3	NaN	10000000	0.000026	Barker	
4	NaN	10000000	NaN	NaN	
5	NaN	10000000	NaN	NaN	
6	NaN	10000000	NaN	NaN	
7	NaN	10000000	NaN	NaN	
8	NaN	10000000	NaN	NaN	
9	NaN	10000000	NaN	NaN	
10	NaN	10000000	NaN	NaN	
11	NaN	10000000	NaN	NaN	
12	Down	10000000	0.000003	NaN	
13	NaN	10000000	NaN	NaN	
14	Up	10000000	0.000004	NaN	
15	NaN	10000000	0.000002	NaN	
16	NaN	10000000	NaN	NaN	
17	Down	10000000	0.000021	NaN	
18	NaN	10000000	NaN	NaN	

19	NaN	10000000	NaN	NaN
\		SUID	radarStatus	radarSignalCenterFreq
0		NaN	0	NaN
1		NaN	0	NaN
2	1c19ef39-f88b-4ff9-a77d-68bc9eb141ef		1	0.0
3	b6e3b197-b0ff-49d2-916c-5bc60df615e4		1	1623597.0
4		NaN	0	NaN
5		NaN	0	NaN
6		NaN	0	NaN
7		NaN	0	NaN
8		NaN	0	NaN
9		NaN	0	NaN
10		NaN	0	NaN
11		NaN	0	NaN
12	6ac0e889-f818-49de-9e7b-40779815a572		1	0.0
13		NaN	0	NaN
14	6d0b9cf4-8efd-4e2c-bc77-139f0c4a8054		1	0.0
15	00c8c6d2-064b-4396-b340-b0ce8d321716		1	-705689.0
16		NaN	0	NaN
17	f9643519-bda0-47e9-9ea0-9745406539c6		1	-1046091.0
18		NaN	0	NaN
19		NaN	0	NaN
		radarSignalStartTime	SNR	NoisePowerdBmPerMHz
0		NaN	NaN	-109
1		NaN	NaN	-109
2	0.008170	10.0		-109
3	0.015718	18.0		-109
4		NaN	NaN	-109
5		NaN	NaN	-109
6		NaN	NaN	-109
7		NaN	NaN	-109
8		NaN	NaN	-109
9		NaN	NaN	-109
10		NaN	NaN	-109
11		NaN	NaN	-109
12	0.018937	16.0		-109
13		NaN	NaN	-109
14	0.020553	14.0		-109
15	0.022690	16.0		-109
16		NaN	NaN	-109
17	0.022535	16.0		-109
18		NaN	NaN	-109
19		NaN	NaN	-109

SNRs Values in the file: [nan 10. 18. 16. 14. 12. 20.]

WAVEFORMS in the file: [nan 'Q3N#3' 'P0N#2' 'Q3N#1' 'P0N#1' 'Q3N#2']

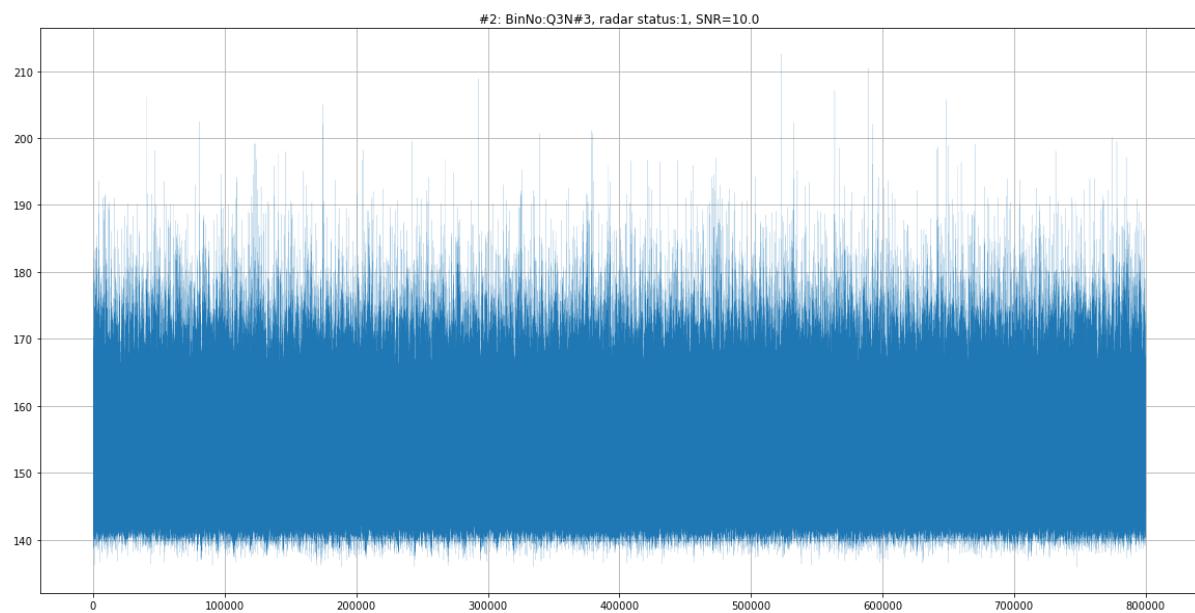
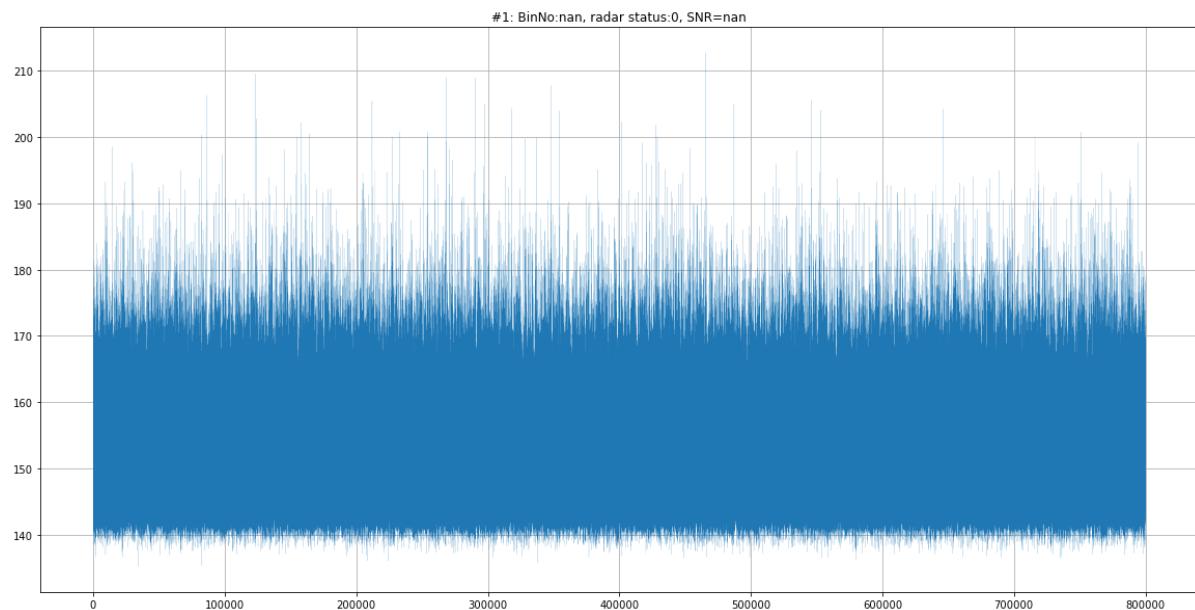
***DONE QUICKLY

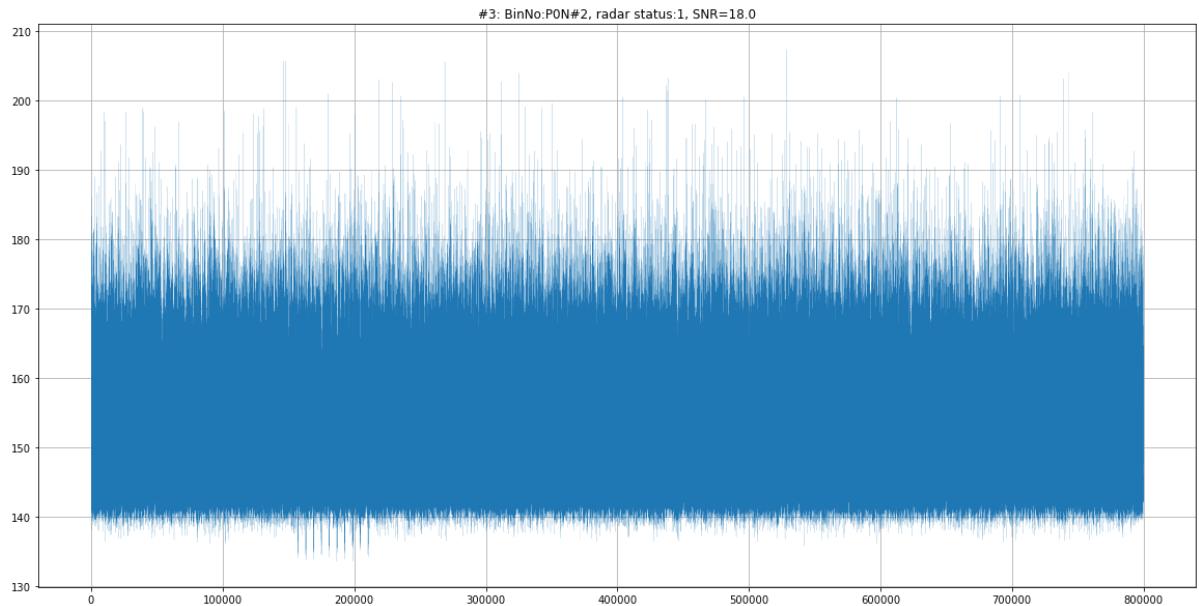
Below we plot three waveforms in the time domain.

The first has only noise, the second has a relatively low SNR and the third a relatively higher SNR. Note that the signal can not be visually detected and it seems that only noise is present.

```
In [7]: noExamples=3
startFrom=1
for sigIndex in range(startFrom,startFrom+noExamples):
    #sigIndex=6 # Matlab index 6
    print('***Doing for sigIndex=', sigIndex)
    plt.figure(figsize=(20,10))
    plt.plot(-10*np.log(np.absolute(subsetSignals[sigIndex][0:-1])),linewidt
h=0.1)
    plt.title('#'+str(sigIndex)+': BinNo:' +str(subsetInfo['BinNo'][sigIndex
])+' , radar status:' +str(subsetInfo['radarStatus'][sigIndex])+', SNR=' +str(sub
setInfo['SNR'][sigIndex]))
    plt.grid()
print('***DONE!')
```

***Doing for sigIndex= 1
***Doing for sigIndex= 2
***Doing for sigIndex= 3
***DONE!





Below we Plot Three Waveforms in the Frequency vs Time Domain.

Again the first has only noise, the second has a relatively low SNR and the third a relatively higher SNR. Note that now the signal is resolved cells of time and frequency the signal can be seen as non-random cell patterns on top of the 'sea of noise'. It is hardly noticeable in the second case (SNR=10 dB), but still it can be seen like a 'cat scratch' o lower left quadrant of the spectrogram. In the case of the third spectrogram, the signal is much easier to see because of the higher SNR and also because the specific waveform used is short and 'shines more' above the noise 'sea level'

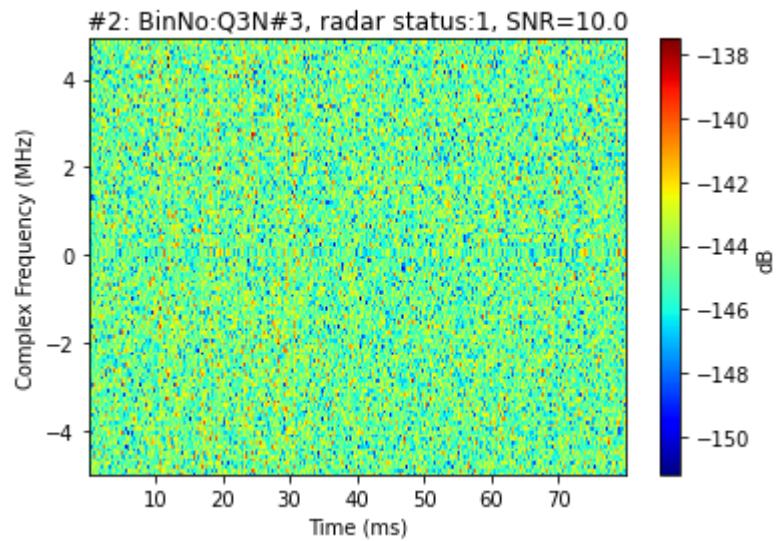
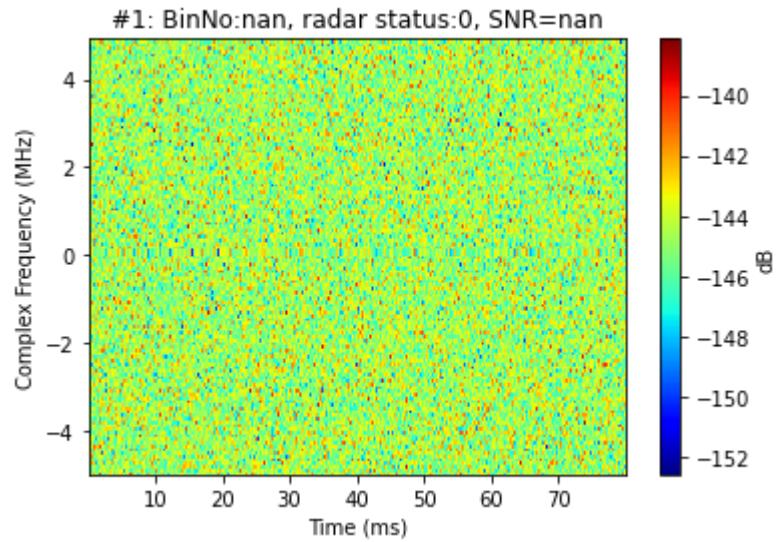
```
In [8]: from scipy import signal
Nfft=128
groupby=16 # no. of consecutive FFTs over which to take max

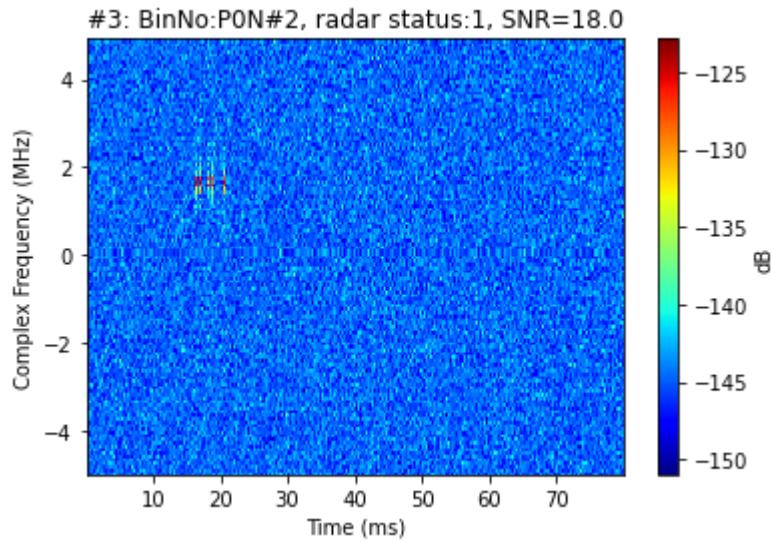
noExamples=3
startFrom=1
for sigIndex in range(startFrom,startFrom+noExamples):
    f, t0, S0 = signal.spectrogram(subsetSignals[sigIndex], fs=10e6, nperseg=Nfft,
t, scaling='spectrum', return_onesided=False)
    print('S0.shape=', S0.shape)
    #
    S0[0:1, :] = S0[1:2, :]
L = S0.shape[1]/groupby
S1 = np.reshape(S0[:, :int(L)*groupby], (Nfft,int(L),groupby))
S = npamax(S1, axis=-1)
t = t0[groupby-1::groupby]

print('S.shape=', S.shape)
#S[0:1, :] = np.mean(S[70:80,:], axis=0)
#S[0:1, :] = S[1:2, :]
fshi = np.fft.fftshift(f/1e6)
Sshi = np.fft.fftshift(10.*np.log10(S), axes=0)
#print('Sshi.shape=', Sshi.shape
#
#fig = plt.figure(figsize=(10,10))
fig = plt.figure()
cax = plt.pcolormesh(t*1e3, fshi, Sshi, cmap='jet')
plt.title('#'+str(sigIndex)+': BinNo:' +str(subsetInfo['BinNo'][sigIndex])+',
radar status:' +str(subsetInfo['radarStatus'][sigIndex])+', SNR=' +str(subsetInfo['SNR'][sigIndex]))
plt.xlabel('Time (ms)')
plt.ylabel('Complex Frequency (MHz)')
fig.colorbar(cax).set_label('dB')

print('***DONE!')
```

```
S0.shape= (128, 7142)
S.shape= (128, 446)
S0.shape= (128, 7142)
S.shape= (128, 446)
S0.shape= (128, 7142)
S.shape= (128, 446)
***DONE!
```





Below we Assess if PCA can Help with the Detection and Classification of Radar Waveforms

As the plots will confirm below, we are hunting for 'anomalies/disturbances' in the non-normalized PC decay curves with the hope that these changes (when compared to the channel with only gaussian noise) will provide some hints that will allow us to detect the presence of the signal and perhaps classify it based on its signature. As we will be able to see below, there are Principal Components (PCs) derived from the Single Value Decomposition (SVD) can potentially give information about the presence waveforms. So some level of classification based on PCs seem to be possible.

In Signal Intelligence, Cybersecurity or Waveform Co-existence Monitoring problems we don't know *a priori* the types of waveform that are present and their exact locations in time and/or frequency. This makes challenging to use **matched filters** techniques without **learning from the waveform data** the information about the signal we are trying to filter match to maximize detection under noise.

One idea (naive perhaps we must add, because we haven't really checked papers and books about it or how to make it more efficient yet, and because we are just trying to explore ML techniques for signals) is to create a time frequency matrix (image) of the signal and use it as an input for ML algorithms. The time frequency analysis provides information about the signal and both domains and this is certainly important information to properly classify a waveform.

In the NIST data base the waveforms can be one of six types [nan 'Q3N#3' 'P0N#2' 'Q3N#1' 'P0N#1' 'Q3N#2'], where 'nan' means that no waveform. These waveforms and the additive gaussian white noise comprise the recorded waveform time series in the files. More information about the waveforms can be found in the NIST documentation referenced above.

Below we do a spectrogram of the signal and do a **maxpool-type reduction** of the data in the time domain in the way suggested by the example program provided by NIST to 'reduce' the signal in the time domain to fewer bins. We have chosen to create 150x150 spectrograms to make them compatible with the VGG16 networks (for future expansion of this project we suggest to try that later).

After that we do an **SVD** of the result matrix to discover the directions of maximum variation of the signal in the time/frequency domain space. As we see below, the first PC (singular value) is the one by two orders of magnitude larger, containing the majority of time/frequency variation. It is in this direction that most of noise and signal variation concentrates in the time frequency space.

More Interestingly, the relative amplitudes of the PCs also change in the presence of a coherent, structured signal and the variations *leak* in the rest of the PCs in different ways depending on the signal signature.

The stem plots below demonstrate that. So we raise the **hypothesis** that the PCs relative distribution can be used to detect the presence of the signal and perhaps even tell what type of waveform is present, which is important to identify the emmitter in some use cases.

Also, the images show interesting features of the waveforms. **The images themselves could be used for classification***. The better the time frequency extraction the easier is to extract identifiable features for classification.

For the current exercise, ***we will focus in using the PCAs for classification with Logistic Regression***, but an obvious extension of this work as mentioned above would to use the images (the time frequency spectrogram matrices) do classification using SVM or Deep Neural Networks (*perhaps using transfer training, which is an intriguing and interesting idea to explore per se*).

NOTE: in this problem we don't scale the data, this is because all the dimensions are already in same units and the relative values of one against the other provide information about the classification. In our case we don't want a fidel reconstruction of signal. We aim for one that stresses some of waveform characteristics that help waveform classification in the presence of the noise (a filter problem, but it does need to be a 'perfect' filter in the sense to reconstruct the best representation of signal in the presence of noise - we want a reconstruction that optimizes classification and our hope/educated guess is that PCA will help with that).

We will then only use the scaling provided by the `signal.spectrogram()` function.

```
In [9]: from scipy import signal
plt.rcParams.update({'figure.max_open_warning': 0})
flag_make_it_150x150=True
pca0 = 1
nrungs_noise = 2
nrungs_signal = 18
Nfft=128
groupby=16 # no. of consecutive FFTs over which to take max
nrungs=nrungs_noise + nrungs_signal
if flag_make_it_150x150:
    Nfft=150
    groupby=40 # no. of consecutive FFTs over which to take max

# [10. 12. 14. 16. 18. 20.]
#IselWaves_signal=np.where(subsetInfo.SNR==10)
#IselWaves_signal=np.where(subsetInfo.SNR==14)
#IselWaves_signal=np.where(subsetInfo.SNR==16)
#IselWaves_signal=np.where(subsetInfo.SNR==18)
IselWaves_signal=np.where(subsetInfo.SNR==20)
IselWaves_noise=np.where(np.isnan(subsetInfo.SNR)) #Noise Only
IselWaves = []
rfix = Nfft//2
idx=1
for I in np.nditer(IselWaves_noise):
    if idx > nrungs_noise: break
    IselWaves.append(int(I))
    idx += 1
idx=1
for I in np.nditer(IselWaves_signal):
    if idx > nrungs_signal: break
    IselWaves.append(int(I))
    idx += 1

idx=1
for I in IselWaves:
    if idx > nrungs: break
    #sigIndex=int(I)
    sigIndex=(I)
    if np.isnan(subsetInfo['SNR'][sigIndex]):
        stem_col = 'silver'
        wavstr = '\n(Gaussian Noise Only)'
    else:
        stem_col = 'coral'
        wavstr = '\n(Gaussian Noise + Signal)'

    print('*****')
    print('***Doing for sigIndex=', sigIndex, 'I=', I)
    #print('...NSD :', subsetInfo['NoisePowerdBmPerMHz'][sigIndex])
    print('...SNR :', subsetInfo['SNR'][sigIndex])
    print('...BinNo :', subsetInfo['BinNo'][sigIndex])

    #
    f, t0, S0 = signal.spectrogram(subsetSignals[sigIndex], fs=10e6, nperseg=Nff
t, scaling='spectrum', return_onesided=False)
    #
    S0[0:1, :] = S0[1:2, :]
```

```

L = S0.shape[1]/groupby
S1 = np.reshape(S0[:, :int(L)*groupby], (Nfft,int(L),groupby))
S = np.amax(S1, axis=-1)
if flag_make_it_150x150: S=S[:,0:S.shape[0]] #Make is 150x150 Square
t = t0[groupby-1::groupby]

print('...S.shape=', S.shape)
fshi = np.fft.fftshift(f/1e6)
Sshi = np.fft.fftshift(10.*np.log10(S), axes=0)

#from sklearn.preprocessing import StandardScaler
#scal = StandardScaler()
#scal.fit_transform(S)
#Sc=scal.transform(S)
#Sshi = np.fft.fftshift(Sc, axes=0)
#print('...Sc.shape=', Sc.shape)

u, s, vh = np.linalg.svd(Sshi, full_matrices=False);
print('...U.shape=', u.shape)
print('...S.shape=', s.shape)
print('...Vh.shape', vh.shape)
#
aux = u.dot(np.diag(s)).dot(vh)
print('...CHECKDIFFMAT (ORIG vs U.diag(S).Vh): ', np.max(np.abs(aux - Sshi)))
fig = plt.figure()
cax = plt.pcolormesh(t*1e3, fshi, Sshi, cmap='jet')
plt.title('Run# '+str(idx)+ ' '+str(subsetInfo['BinNo'][sigIndex])+' SPECTROGRAM(All PCs)'+wavstr)
plt.xlabel('Time (ms)')
plt.ylabel('Complex Frequency (MHz)')
fig.colorbar(cax).set_label('dB')
#
fig = plt.figure()
scp = np.copy(s)
scp[0:0+1] = 0; #Knocks 2nd PC
aux = u.dot(np.diag(scp)).dot(vh)
aux[rfix:rfix+2,:] = aux[rfix-2:rfix,:]
#Workaround. Orig data has issues @0Hz (?!). Not big prob 4us
cax = plt.pcolormesh(t*1e3, fshi, aux, cmap='jet')
plt.title('Run# '+str(idx)+ ' '+str(subsetInfo['BinNo'][sigIndex])+' SPECTROGRAM (No PC1)'+wavstr)
plt.xlabel('Time (ms)')
plt.ylabel('Complex Frequency (MHz)')
fig.colorbar(cax).set_label('dB')
#
fig = plt.figure()
scp[1:1+1] = 0; #Knocks Off 1st PC Also
aux = u.dot(np.diag(scp)).dot(vh)
aux[rfix:rfix+2,:] = aux[rfix-2:rfix,:]
#Workaround. Orig data has issues @0Hz (?!). Not big prob 4us
cax = plt.pcolormesh(t*1e3, fshi, aux, cmap='jet')
plt.title('Run# '+str(idx)+ ' '+str(subsetInfo['BinNo'][sigIndex])+' SPECTROGRAM (No PCs[1,2])'+wavstr)
plt.xlabel('Time (ms)')
plt.ylabel('Complex Frequency (MHz)')
fig.colorbar(cax).set_label('dB')

```

```

#
fig = plt.figure()
scp = np.copy(s)
scp[0:0+1] = 0; scp[8:] = 0; #Also 1st and all PCs after 8th
aux = u.dot(np.diag(scp)).dot(vh)
aux[rfix:rfix+2,:]=aux[rfix-2:rfix,:] #Workaround. Orig data has issues @0Hz
(?!). Not big prob 4us
cax = plt.pcolormesh(t*1e3, fshi, aux, cmap='jet')
plt.title('Run# '+str(idx)+ ' '+str(subsetInfo['BinNo'][sigIndex])+' SPECTROGRAM (Only Pcs[2 to 8])'+wavstr)
plt.xlabel('Time (ms)')
plt.ylabel('Complex Frequency (MHz)')
fig.colorbar(cax).set_label('dB')
#
fig = plt.figure()
scp = np.copy(s)
scp[0:0+1] = 0; scp[3:] = 0; #Now only 2nd and 3rd PC
aux = u.dot(np.diag(scp)).dot(vh)
aux[rfix:rfix+2,:]=aux[rfix-2:rfix,:] #Workaround. Orig data has issues @0Hz
(?!). Not big prob 4us
cax = plt.pcolormesh(t*1e3, fshi, aux, cmap='jet')
plt.title('Run# '+str(idx)+ ' '+str(subsetInfo['BinNo'][sigIndex])+' SPECTROGRAM (Only Pcs[2,3])'+wavstr)
plt.xlabel('Time (ms)')
plt.ylabel('Complex Frequency (MHz)')
fig.colorbar(cax).set_label('dB')
#
#aux = 100*np.sort(s)[::-1]/np.sum(s)
aux = s/np.sum(s)
print('...S[0:10] (NORMALIZED to SUM to 100%) = \n', aux[0:10])
plt.figure(figsize=(10,8))
xlab = np.arange(pca0, 50+pca0, 1)
plt.stem(xlab, aux[0+pca0:50+pca0], stem_col, use_line_collection=True)
plt.title('Run# '+str(idx)+ ' '+str(subsetInfo['BinNo'][sigIndex])+' PCs (50 largest, first/largest Not Shown). \n NOTE: Plot Normalized: All PCs add to 1'+wavstr)
pch='PCs indexes sorted by magnitude starting from PC='+str(pca0)
plt.xlabel(pch)
plt.ylabel('Relative Strength PC (PC Sum Normalized to 1)');
plt.figure(figsize=(10,8))
lam = s**2
PoV = np.log(np.cumsum(lam)/np.sum(lam))*10
plt.plot(np.arange(1,PoV.shape[0]+1), PoV, '-b')
plt.grid()
plt.xlabel('Number of PCs', fontsize=16)
plt.ylabel('log10(PoV)', fontsize=16)
plt.xlim((1,PoV.shape[0]+1))

#print('...%02d'%sigIndex, ':', 'Min =', 10.*np.Log10(min(S.flat)), '\n.....Max =', 10.*np.Log10(max(S.flat)))
idx += 1;
# end of for()

print('***DONE!')

```

```
***-----
***Doing for sigIndex= 0 I= 0
...SNR   : nan
...BinNo : nan
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
...CHECKDIFFMAT (ORIG vs U.diag(S).Vh):  2.3021584638627246e-12
...S[0:10] (NORMALIZED to SUM to 100%) =
[0.91983785 0.00127481 0.00122507 0.00119239 0.001175  0.00116176
 0.00114517 0.00113336 0.00112209 0.00109473]
***-----
***Doing for sigIndex= 1 I= 1
...SNR   : nan
...BinNo : nan
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
...CHECKDIFFMAT (ORIG vs U.diag(S).Vh):  1.9895196601282805e-12
...S[0:10] (NORMALIZED to SUM to 100%) =
[0.92019345 0.00130539 0.00124545 0.00123469 0.00118239 0.00117644
 0.00113607 0.00112823 0.00111364 0.00110824]
***-----
***Doing for sigIndex= 35 I= 35
...SNR   : 20.0
...BinNo : P0N#2
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
...CHECKDIFFMAT (ORIG vs U.diag(S).Vh):  6.394884621840902e-12
...S[0:10] (NORMALIZED to SUM to 100%) =
[0.9149646 0.00553405 0.00127679 0.00125894 0.00121393 0.00118047
 0.00116793 0.00116124 0.0011562 0.00113433]
***-----
***Doing for sigIndex= 58 I= 58
...SNR   : 20.0
...BinNo : Q3N#3
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
...CHECKDIFFMAT (ORIG vs U.diag(S).Vh):  2.9274360713316128e-12
...S[0:10] (NORMALIZED to SUM to 100%) =
[0.91477053 0.00274237 0.00259661 0.00142077 0.00139571 0.00132032
 0.00126642 0.00120036 0.0011734 0.00115988]
***-----
***Doing for sigIndex= 65 I= 65
...SNR   : 20.0
...BinNo : P0N#1
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
...CHECKDIFFMAT (ORIG vs U.diag(S).Vh):  2.4726887204451486e-12
```

```

...S[0:10] (NORMALIZED to SUM to 100%) =
[0.91765343 0.00231691 0.00125611 0.00122822 0.00120347 0.00119546
0.00117893 0.00116042 0.00114171 0.00111813]
***-----
***Doing for sigIndex= 68 I= 68
...SNR : 20.0
...BinNo : Q3N#1
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
...CHECKDIFFMAT (ORIG vs U.diag(S).Vh): 2.1032064978498966e-12
...S[0:10] (NORMALIZED to SUM to 100%) =
[0.91388013 0.00394099 0.00156884 0.0014213 0.00131497 0.00127931
0.00123585 0.00119945 0.00116732 0.00115151]
***-----
***Doing for sigIndex= 91 I= 91
...SNR : 20.0
...BinNo : P0N#2
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
...CHECKDIFFMAT (ORIG vs U.diag(S).Vh): 4.490630090003833e-12
...S[0:10] (NORMALIZED to SUM to 100%) =
[0.91583381 0.00458533 0.00121935 0.00120956 0.0011942 0.00118662
0.00115026 0.00113951 0.00113306 0.00112921]
***-----
***Doing for sigIndex= 111 I= 111
...SNR : 20.0
...BinNo : Q3N#2
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
...CHECKDIFFMAT (ORIG vs U.diag(S).Vh): 2.4726887204451486e-12
...S[0:10] (NORMALIZED to SUM to 100%) =
[0.91877696 0.00202446 0.00127082 0.00124709 0.0012271 0.00119706
0.00117838 0.00115496 0.00114193 0.00112805]
***-----
***Doing for sigIndex= 114 I= 114
...SNR : 20.0
...BinNo : P0N#1
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
...CHECKDIFFMAT (ORIG vs U.diag(S).Vh): 1.8758328224066645e-12
...S[0:10] (NORMALIZED to SUM to 100%) =
[0.91667905 0.00342856 0.00127163 0.00125562 0.00122589 0.00120123
0.00118848 0.00117614 0.0011548 0.00113471]
***-----
***Doing for sigIndex= 115 I= 115
...SNR : 20.0
...BinNo : P0N#1
...S.shape= (150, 150)
...U.shape= (150, 150)

```

```

....S.shape= (150, )
....Vh.shape (150, 150)
....CHECKDIFFMAT (ORIG vs U.diag(S).Vh):  1.9610979506978765e-12
....S[0:10] (NORMALIZED to SUM to 100%) =
[0.91587187 0.00375527 0.0012744 0.00126157 0.00121259 0.00119711
 0.00115284 0.00112505 0.00112126 0.00111307]
***-----
***Doing for sigIndex= 117 I= 117
...SNR : 20.0
...BinNo : Q3N#2
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
....CHECKDIFFMAT (ORIG vs U.diag(S).Vh):  4.320099833421409e-12
....S[0:10] (NORMALIZED to SUM to 100%) =
[0.91427553 0.00551098 0.00147408 0.00126767 0.00121866 0.00118554
 0.00116914 0.00116462 0.00113491 0.00111884]
***-----
***Doing for sigIndex= 119 I= 119
...SNR : 20.0
...BinNo : P0N#1
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
....CHECKDIFFMAT (ORIG vs U.diag(S).Vh):  2.1316282072803006e-12
....S[0:10] (NORMALIZED to SUM to 100%) =
[0.91801786 0.00227813 0.00129252 0.00125248 0.00122077 0.0012131
 0.00117509 0.00115719 0.00113627 0.00112587]
***-----
***Doing for sigIndex= 124 I= 124
...SNR : 20.0
...BinNo : Q3N#2
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
....CHECKDIFFMAT (ORIG vs U.diag(S).Vh):  1.9042545318370685e-12
....S[0:10] (NORMALIZED to SUM to 100%) =
[0.91772436 0.00263085 0.00123671 0.00123338 0.00121667 0.00119499
 0.00117821 0.00115177 0.00114179 0.0011146 ]
***-----
***Doing for sigIndex= 125 I= 125
...SNR : 20.0
...BinNo : Q3N#1
...S.shape= (150, 150)
...U.shape= (150, 150)
...S.shape= (150,)
...Vh.shape (150, 150)
....CHECKDIFFMAT (ORIG vs U.diag(S).Vh):  2.2168933355715126e-12
....S[0:10] (NORMALIZED to SUM to 100%) =
[0.91188267 0.00824157 0.00129084 0.00126251 0.00122086 0.00120967
 0.00118513 0.00116208 0.0011385 0.00110954]
***-----
***Doing for sigIndex= 139 I= 139
...SNR : 20.0

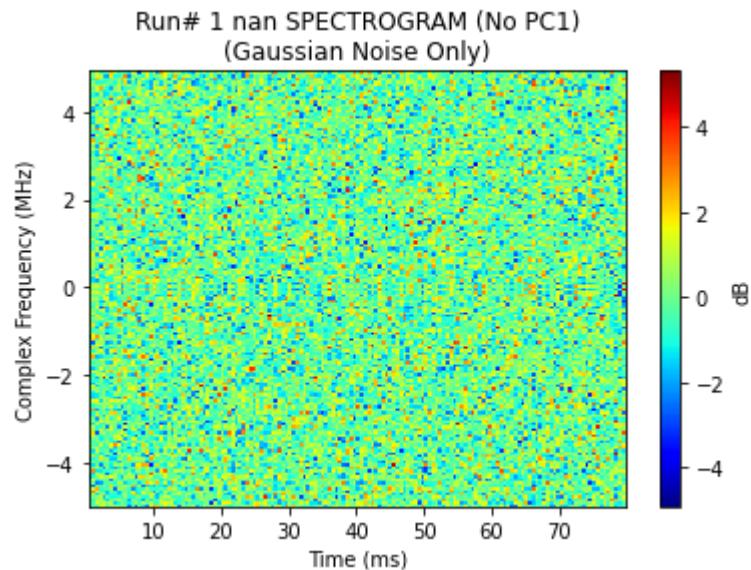
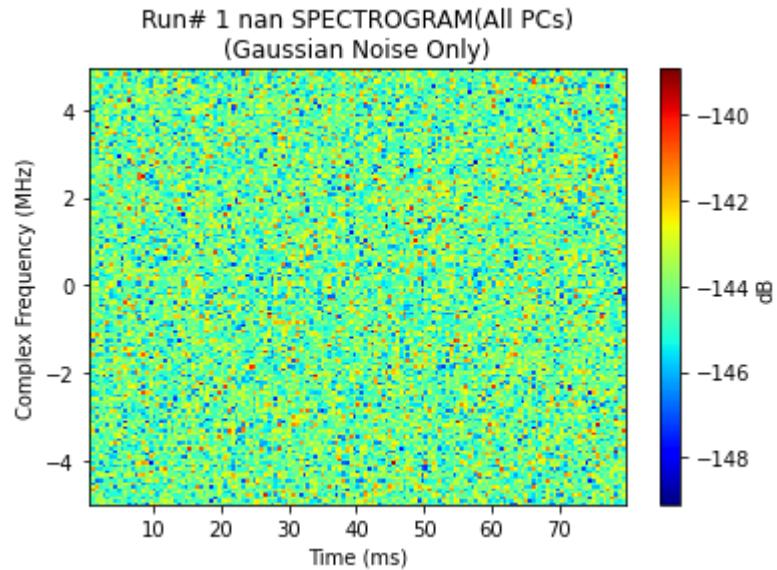
```

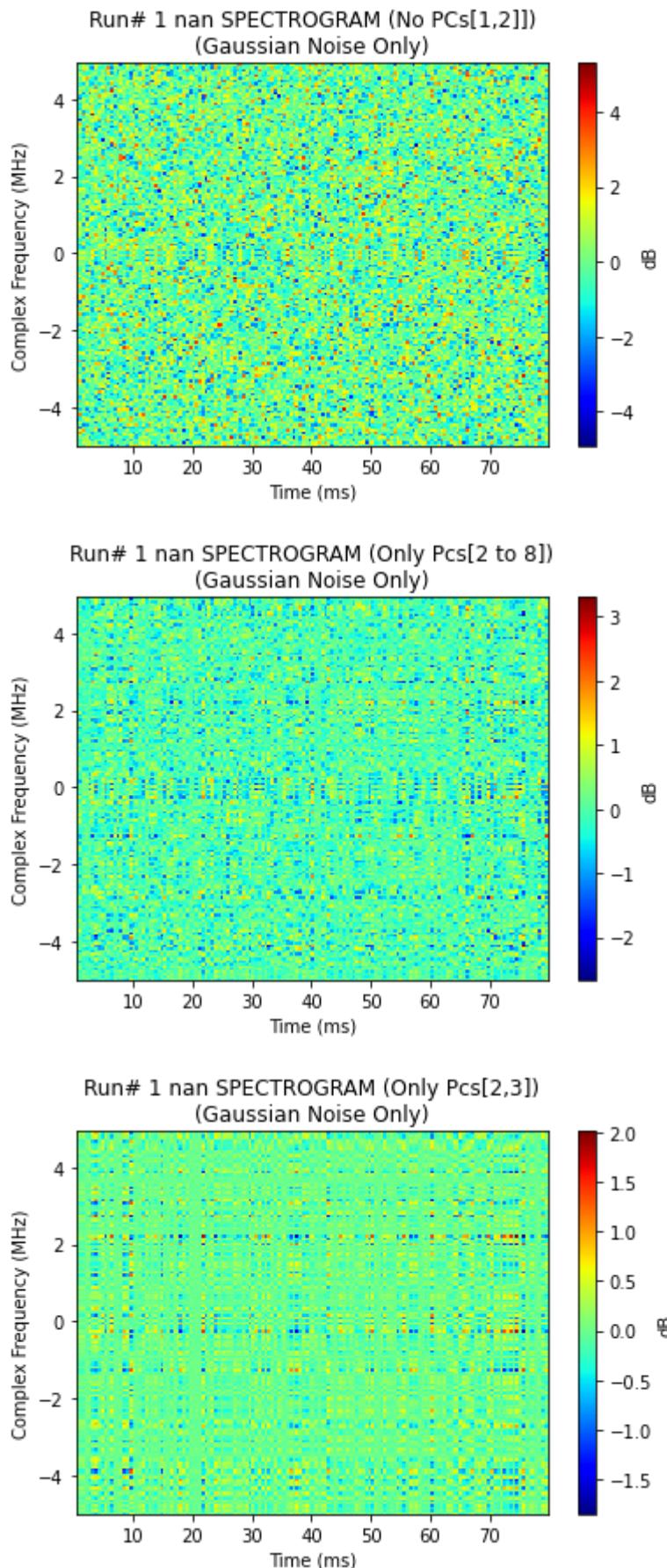
```

....BinNo : P0N#1
....S.shape= (150, 150)
....U.shape= (150, 150)
....S.shape= (150,)
....Vh.shape (150, 150)
....CHECKDIFFMAT (ORIG vs U.diag(S).Vh): 2.0747847884194925e-12
....S[0:10] (NORMALIZED to SUM to 100%) =
[0.91678175 0.00352307 0.00124714 0.00123706 0.00120911 0.0011934
 0.00116897 0.00114408 0.00113639 0.00111789]
***-----
***Doing for sigIndex= 141 I= 141
...SNR : 20.0
...BinNo : P0N#2
....S.shape= (150, 150)
....U.shape= (150, 150)
....S.shape= (150,)
....Vh.shape (150, 150)
....CHECKDIFFMAT (ORIG vs U.diag(S).Vh): 2.2737367544323206e-12
....S[0:10] (NORMALIZED to SUM to 100%) =
[0.91503501 0.00518061 0.00127501 0.00122063 0.00120676 0.00117732
 0.00116205 0.00114086 0.0011359 0.00111901]
***-----
***Doing for sigIndex= 147 I= 147
...SNR : 20.0
...BinNo : Q3N#3
....S.shape= (150, 150)
....U.shape= (150, 150)
....S.shape= (150,)
....Vh.shape (150, 150)
....CHECKDIFFMAT (ORIG vs U.diag(S).Vh): 2.1884716261411086e-12
....S[0:10] (NORMALIZED to SUM to 100%) =
[0.91571179 0.00187964 0.00181203 0.00138379 0.0012718 0.00125069
 0.0012272 0.00119967 0.00118466 0.00116973]
***-----
***Doing for sigIndex= 167 I= 167
...SNR : 20.0
...BinNo : Q3N#1
....S.shape= (150, 150)
....U.shape= (150, 150)
....S.shape= (150,)
....Vh.shape (150, 150)
....CHECKDIFFMAT (ORIG vs U.diag(S).Vh): 5.286437954055145e-12
....S[0:10] (NORMALIZED to SUM to 100%) =
[0.91409672 0.0046742 0.00150484 0.00130632 0.00126942 0.00120999
 0.00119494 0.00117879 0.00116651 0.00116493]
***-----
***Doing for sigIndex= 178 I= 178
...SNR : 20.0
...BinNo : Q3N#3
....S.shape= (150, 150)
....U.shape= (150, 150)
....S.shape= (150,)
....Vh.shape (150, 150)
....CHECKDIFFMAT (ORIG vs U.diag(S).Vh): 2.1316282072803006e-12
....S[0:10] (NORMALIZED to SUM to 100%) =
[0.91731228 0.00148496 0.00144127 0.00129005 0.00127702 0.0012451
 0.00121459 0.00118883 0.00116097 0.00113436]

```

```
***-----  
***Doing for sigIndex= 183 I= 183  
...SNR : 20.0  
...BinNo : Q3N#2  
...S.shape= (150, 150)  
...U.shape= (150, 150)  
...S.shape= (150,)  
...Vh.shape (150, 150)  
...CHECKDIFFMAT (ORIG vs U.diag(S).Vh): 2.2168933355715126e-12  
...S[0:10] (NORMALIZED to SUM to 100%) =  
[0.91819514 0.00143753 0.00135515 0.00127837 0.00125573 0.00122595  
0.00120015 0.00117571 0.00116309 0.00113568]  
***DONE!
```

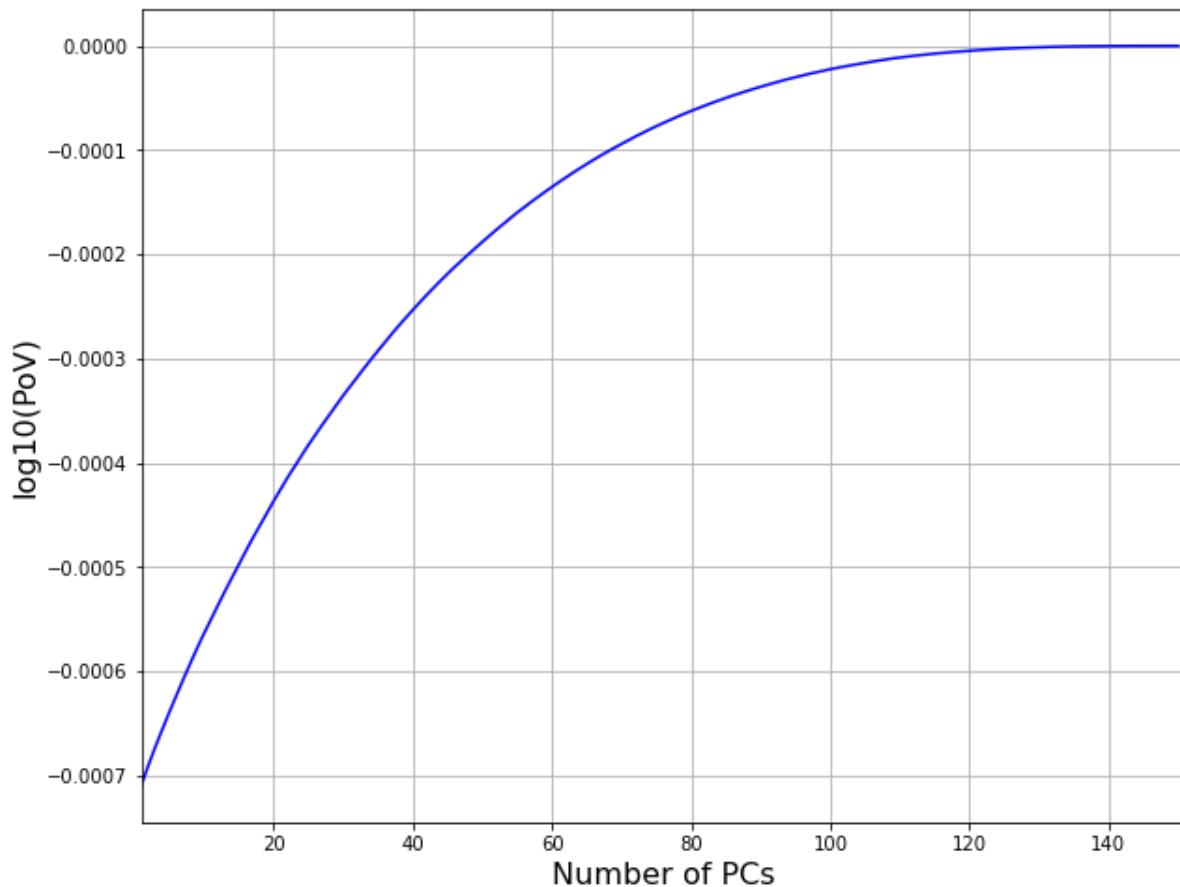
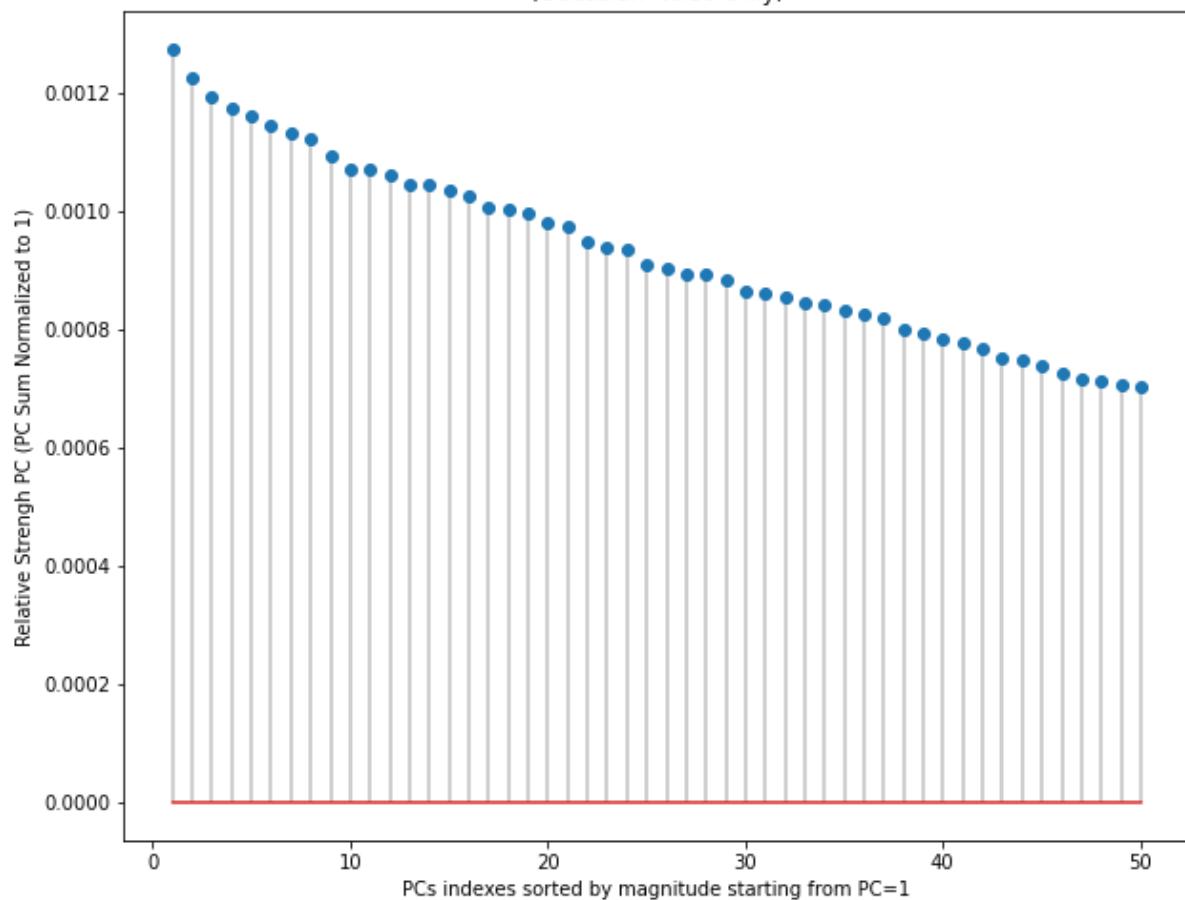


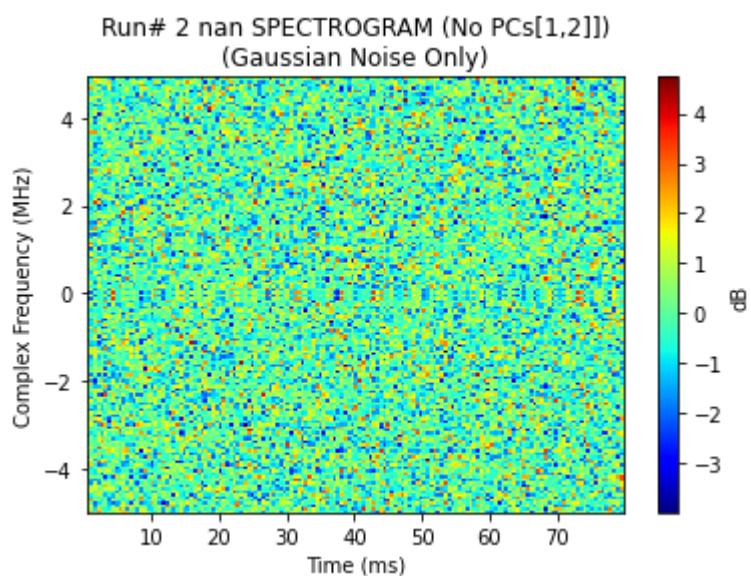
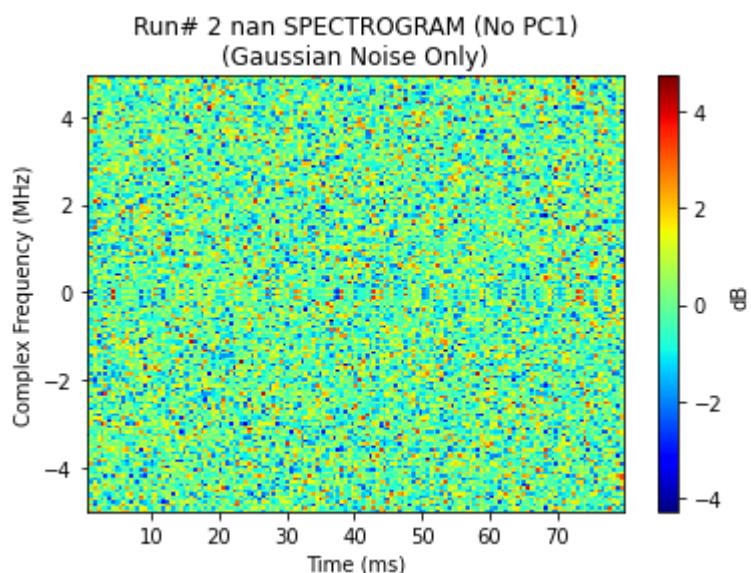
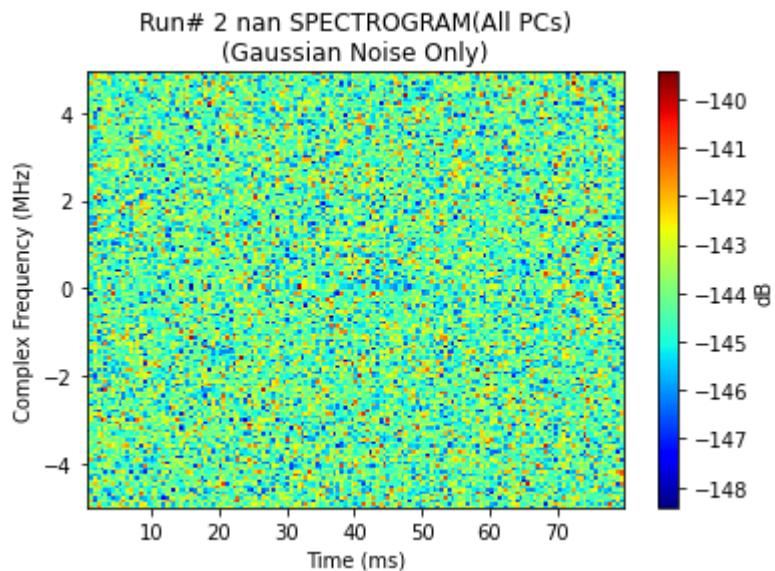


Run# 1 nan PCs (50 largest, first/largest Not Shown).

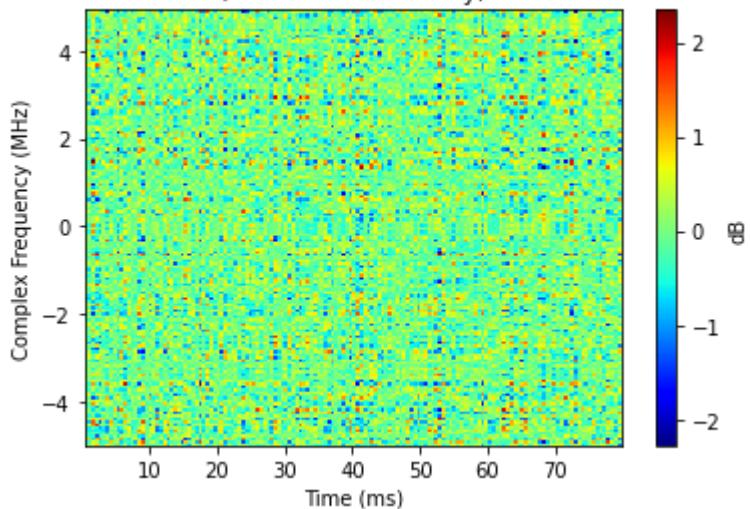
NOTE: Plot Normalized: All PCs add to 1

(Gaussian Noise Only)

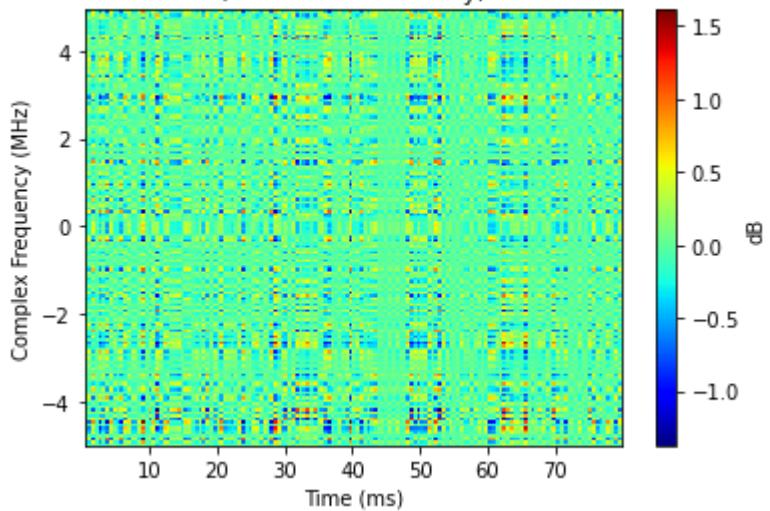




Run# 2 nan SPECTROGRAM (Only Pcs[2 to 8])
(Gaussian Noise Only)



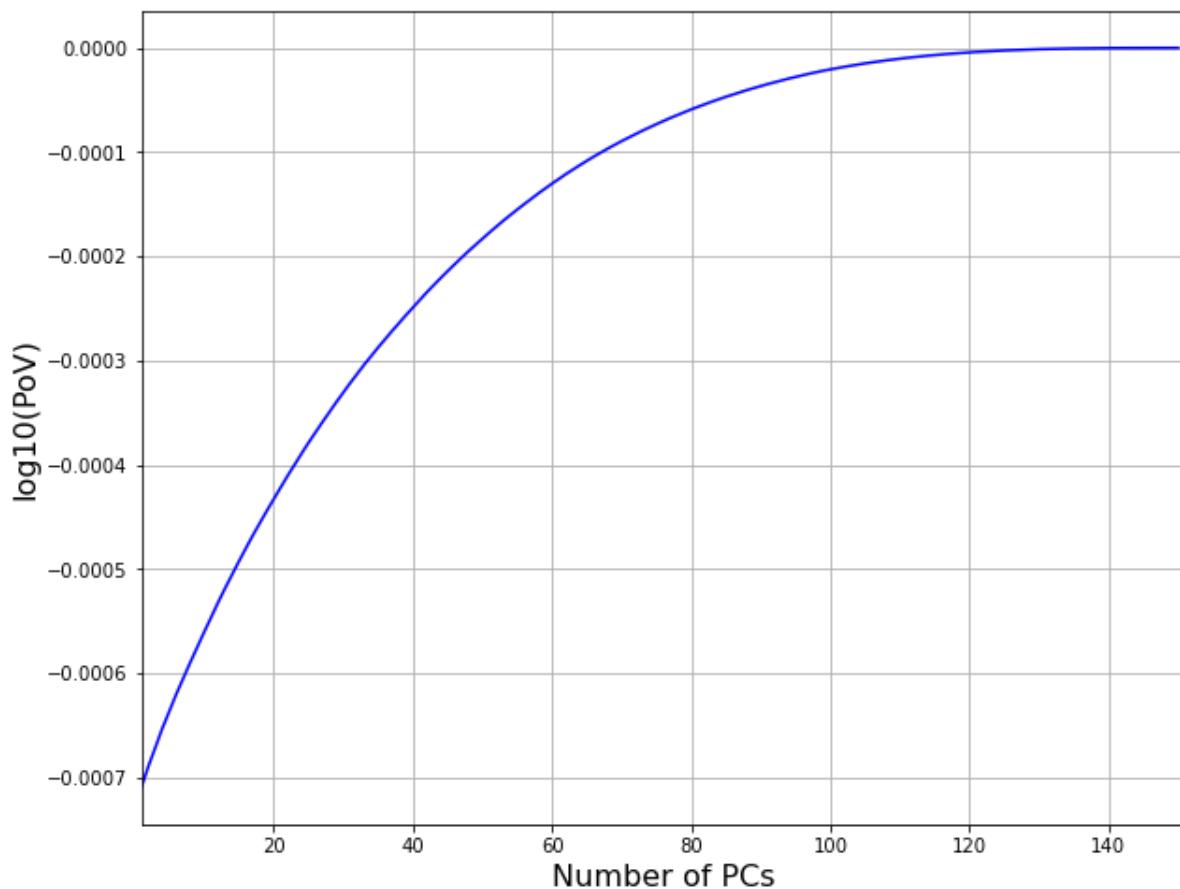
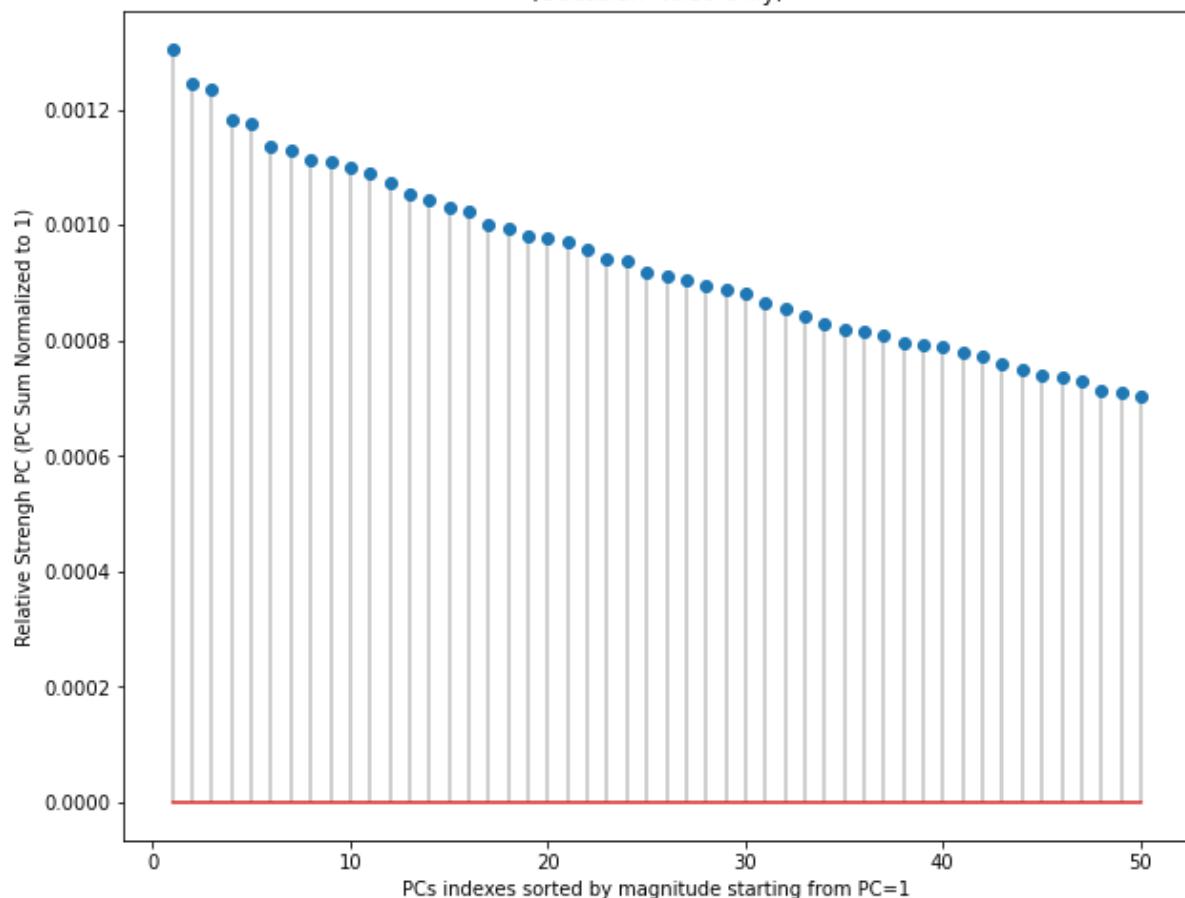
Run# 2 nan SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise Only)

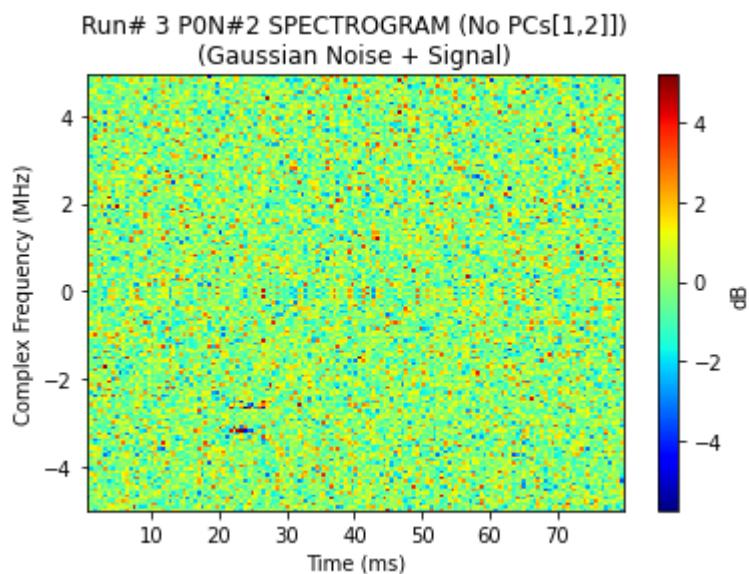
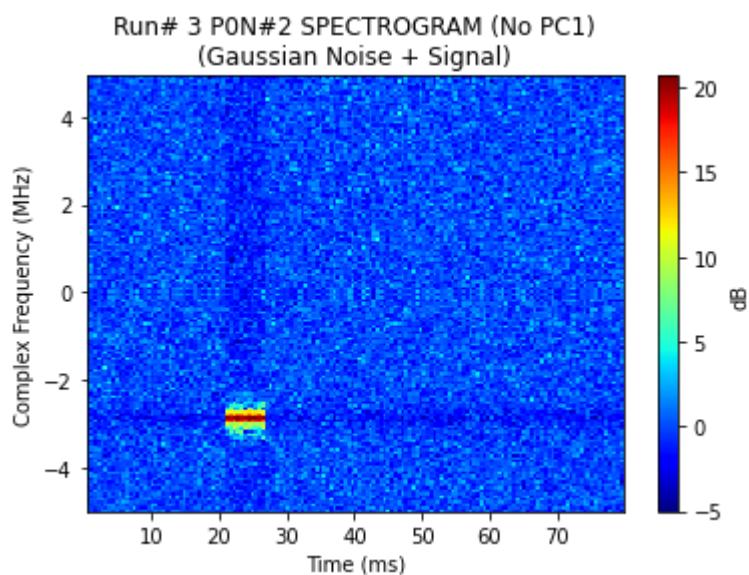
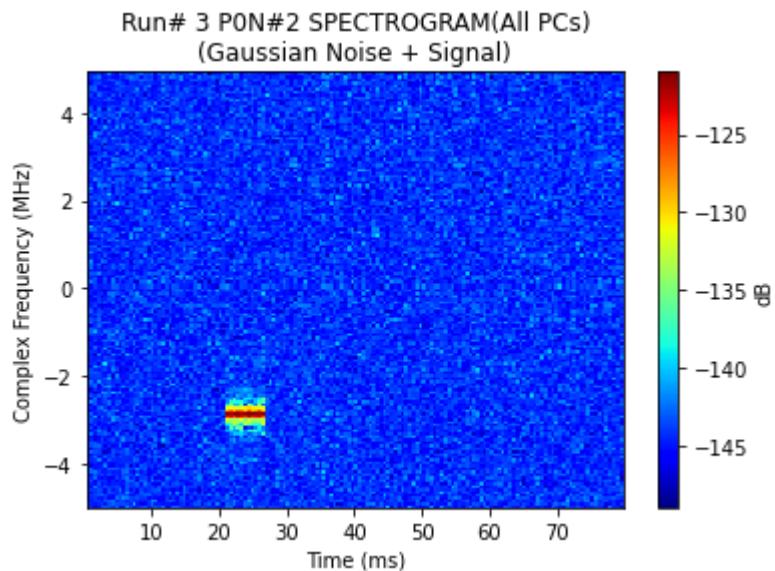


Run# 2 nan PCs (50 largest, first/largest Not Shown).

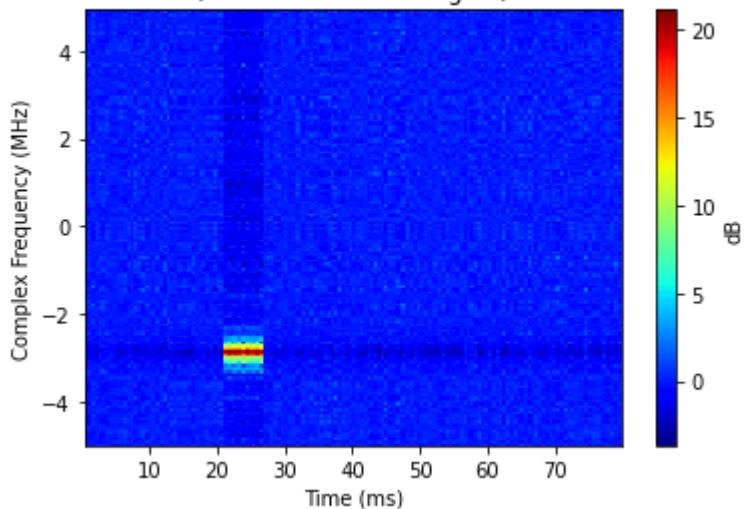
NOTE: Plot Normalized: All PCs add to 1

(Gaussian Noise Only)

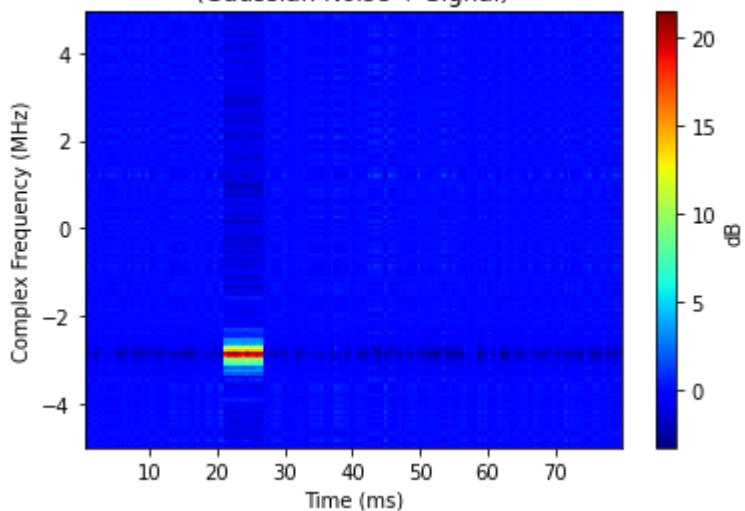




Run# 3 PON#2 SPECTROGRAM (Only Pcs[2 to 8])
(Gaussian Noise + Signal)



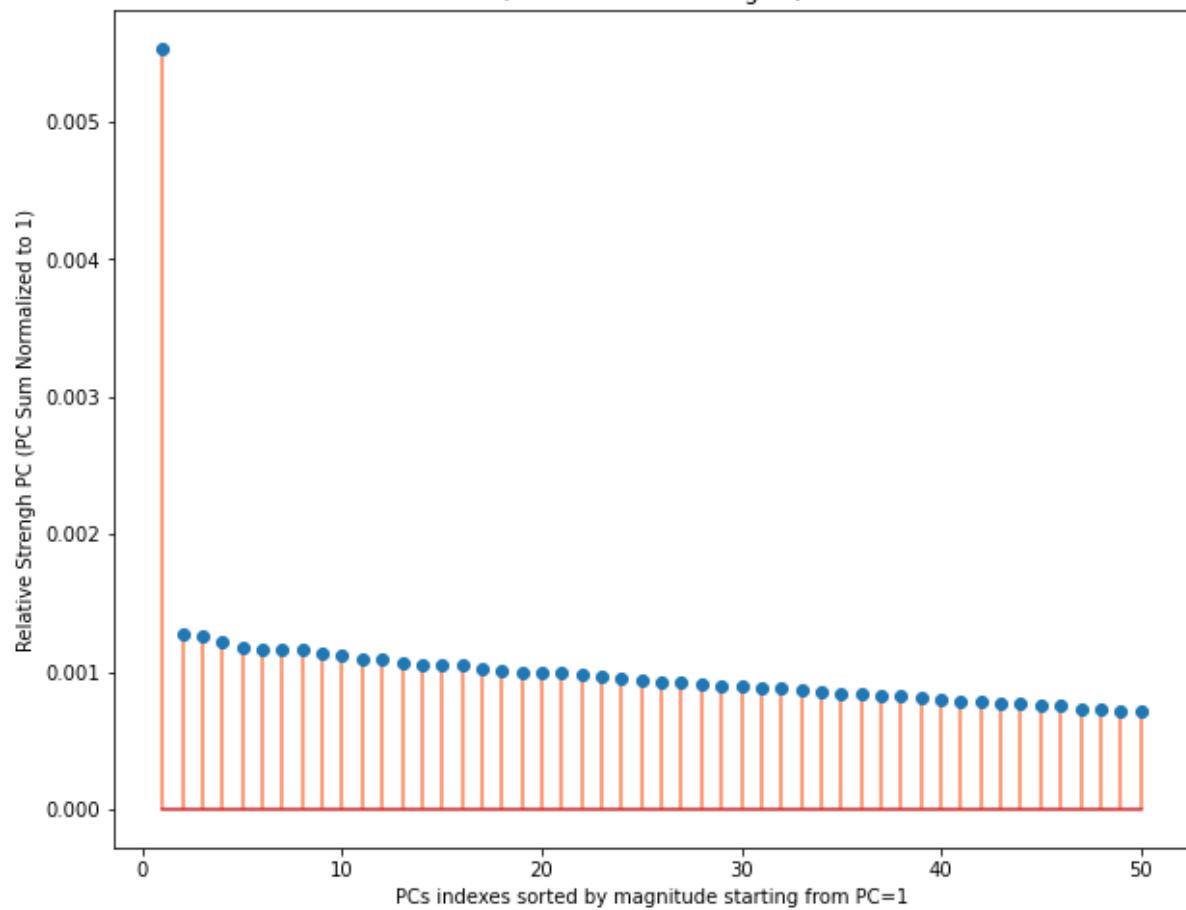
Run# 3 PON#2 SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise + Signal)

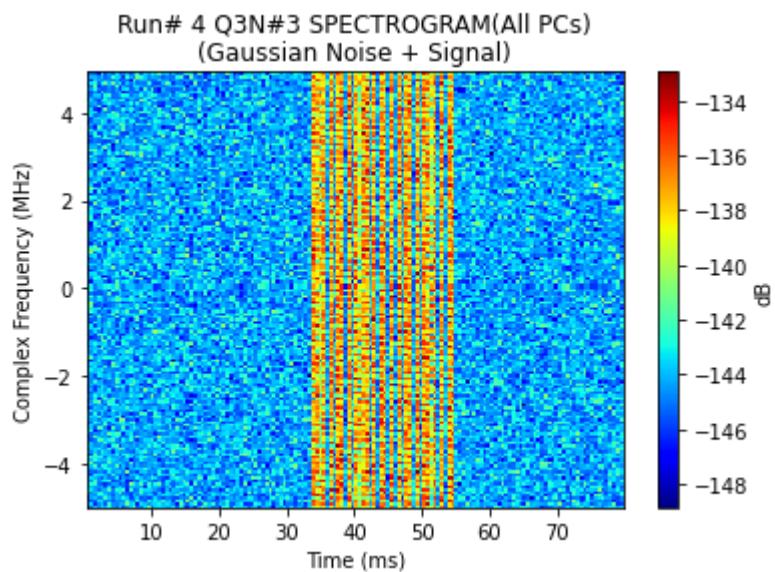
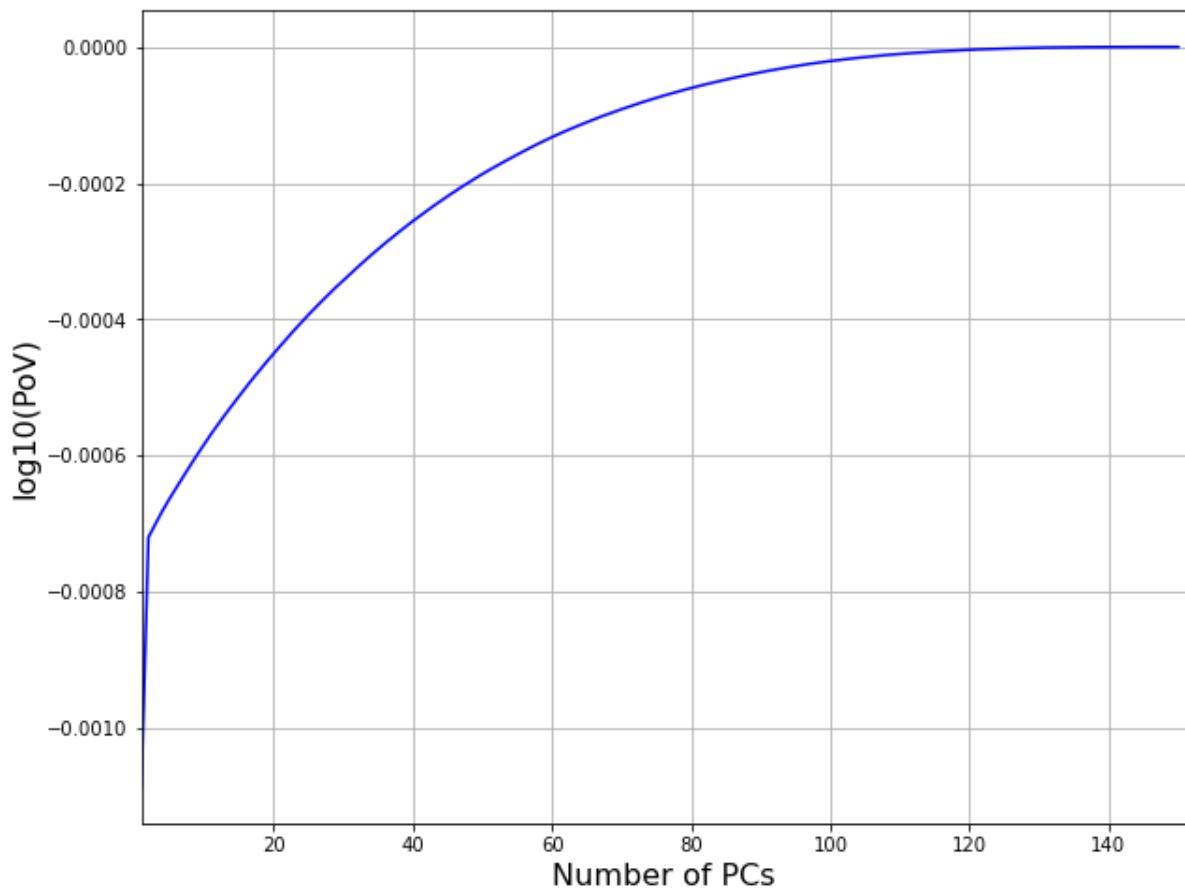


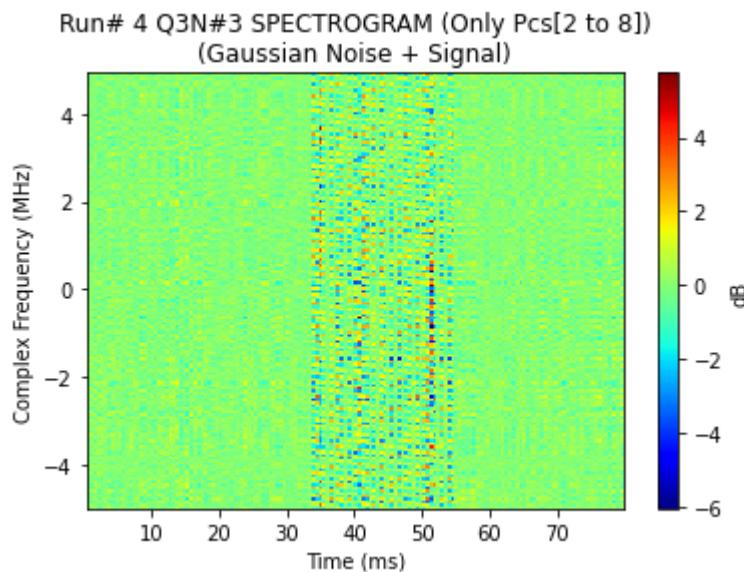
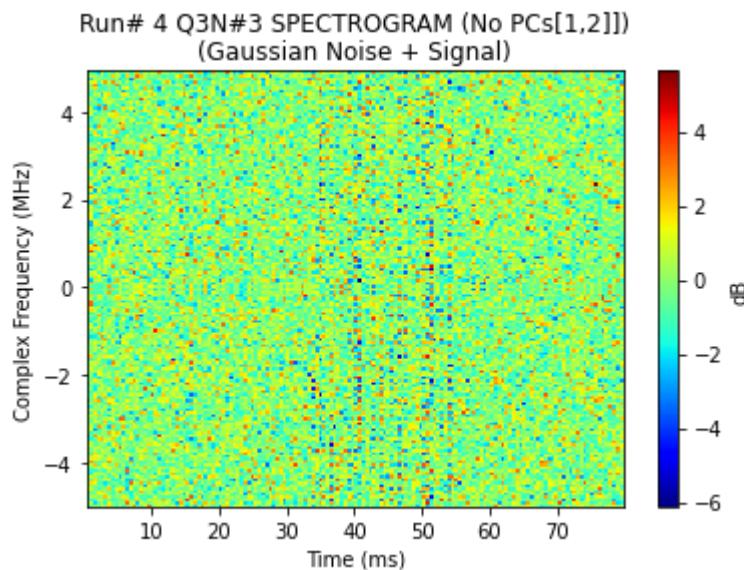
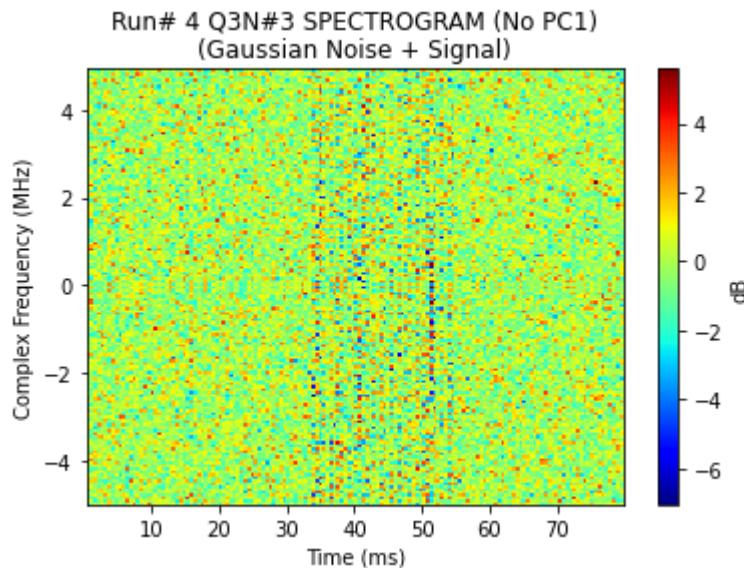
Run# 3 PON#2 PCs (50 largest, first/largest Not Shown).

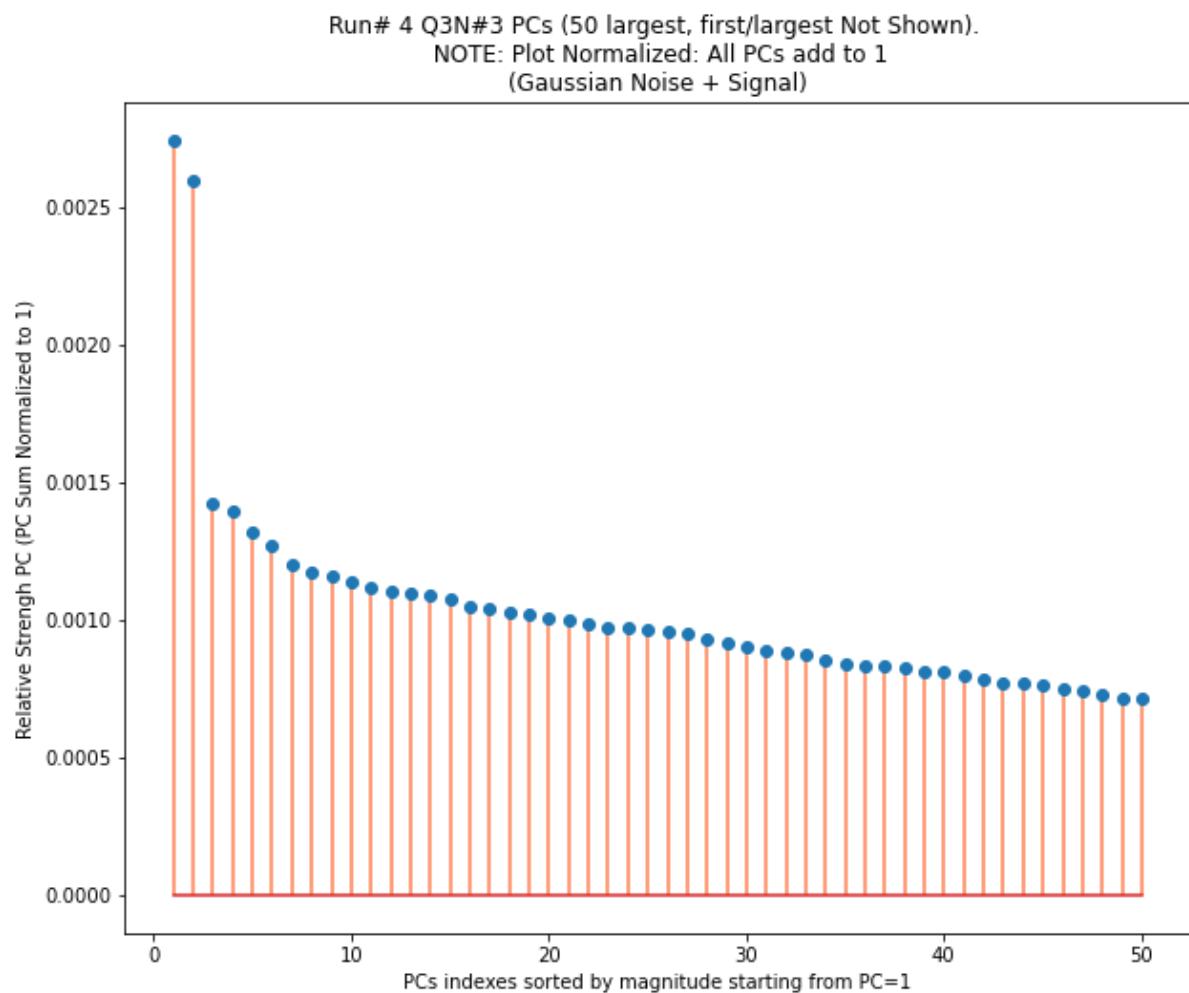
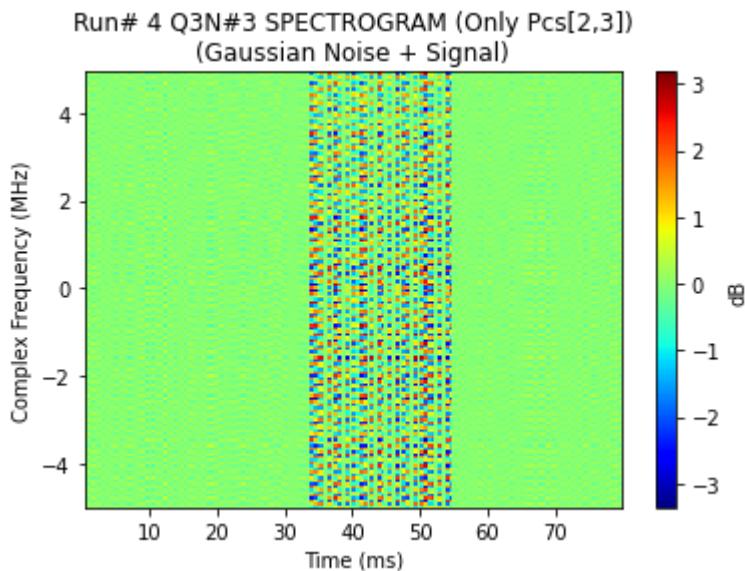
NOTE: Plot Normalized: All PCs add to 1

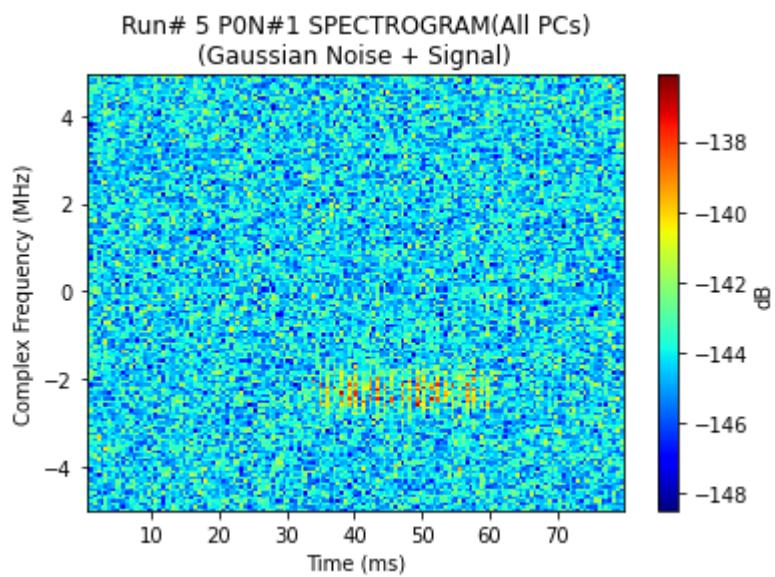
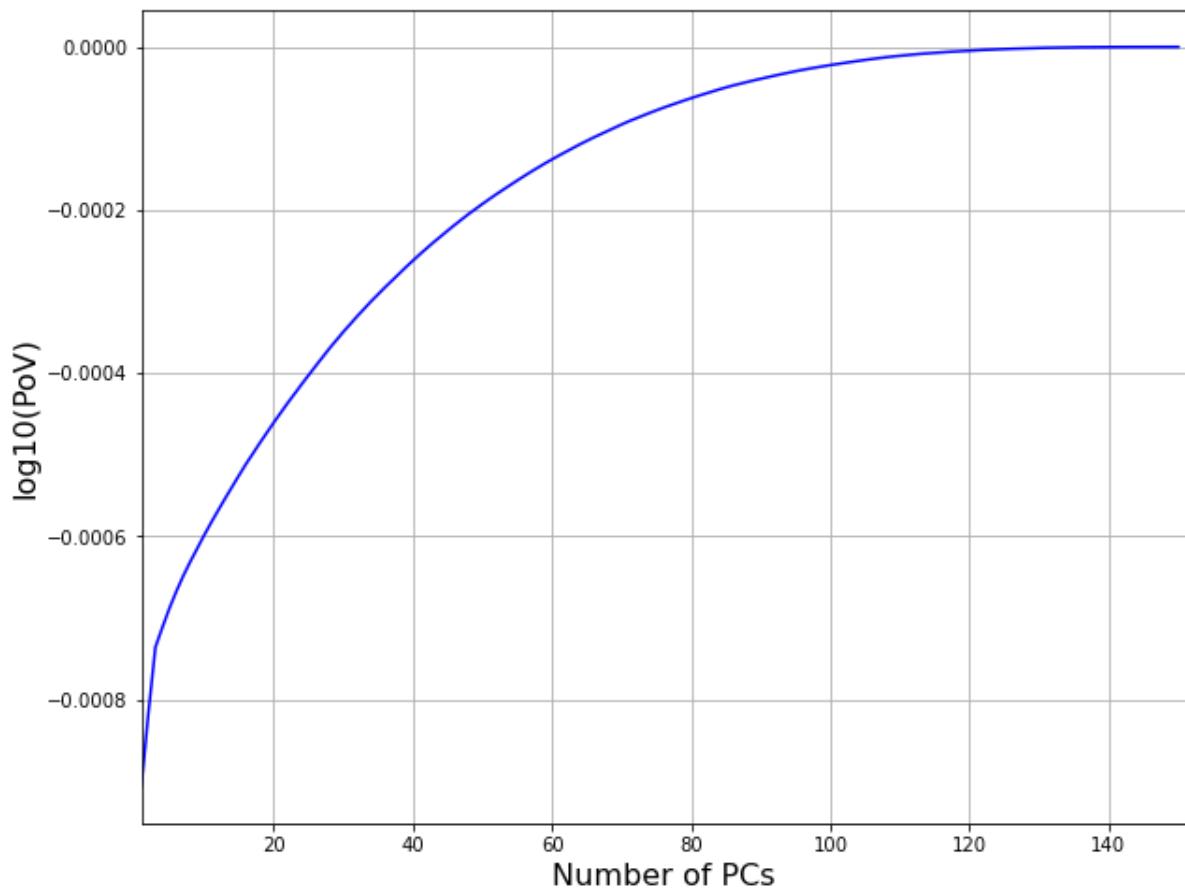
(Gaussian Noise + Signal)

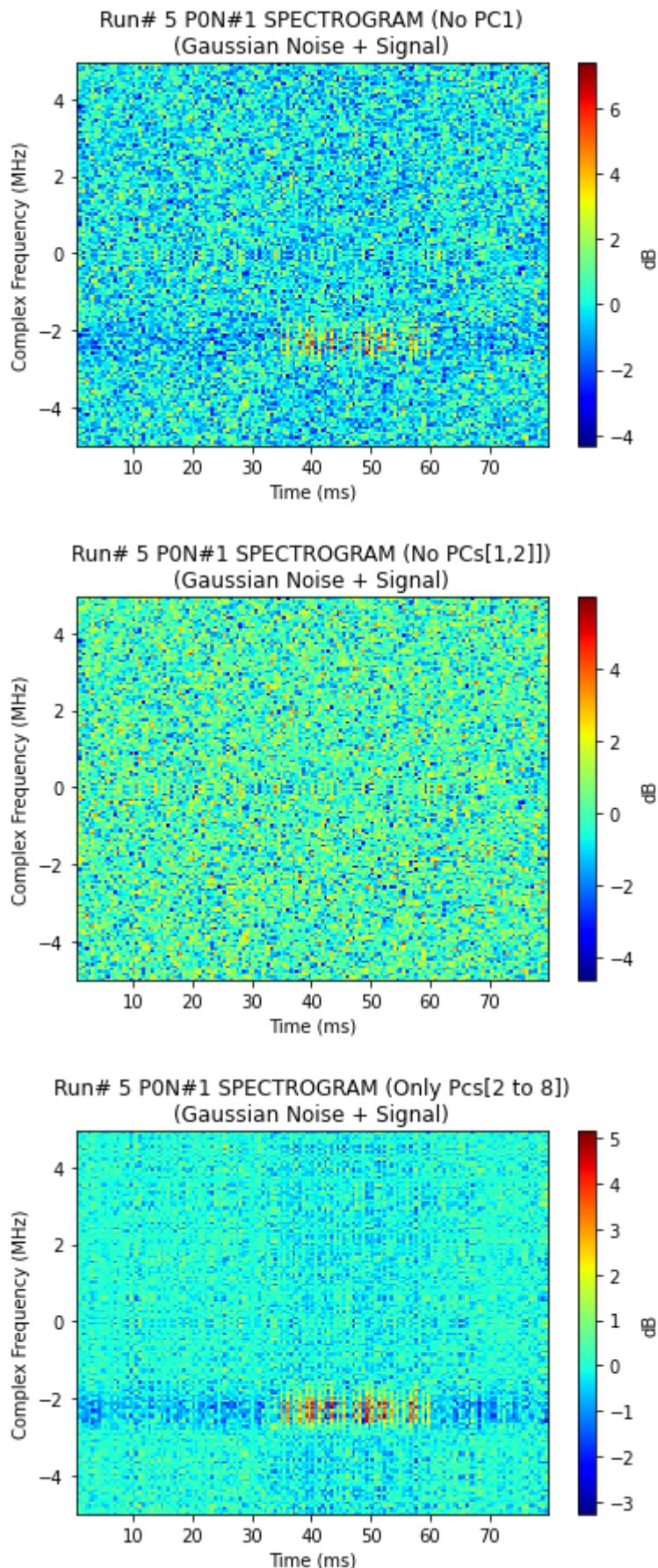




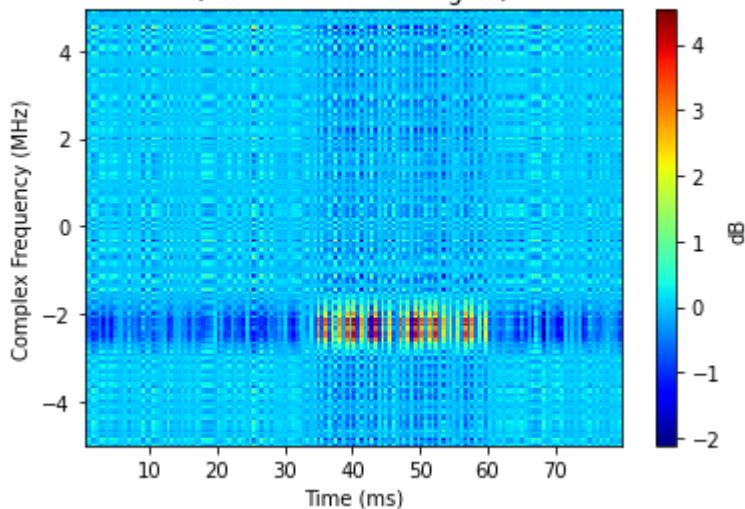




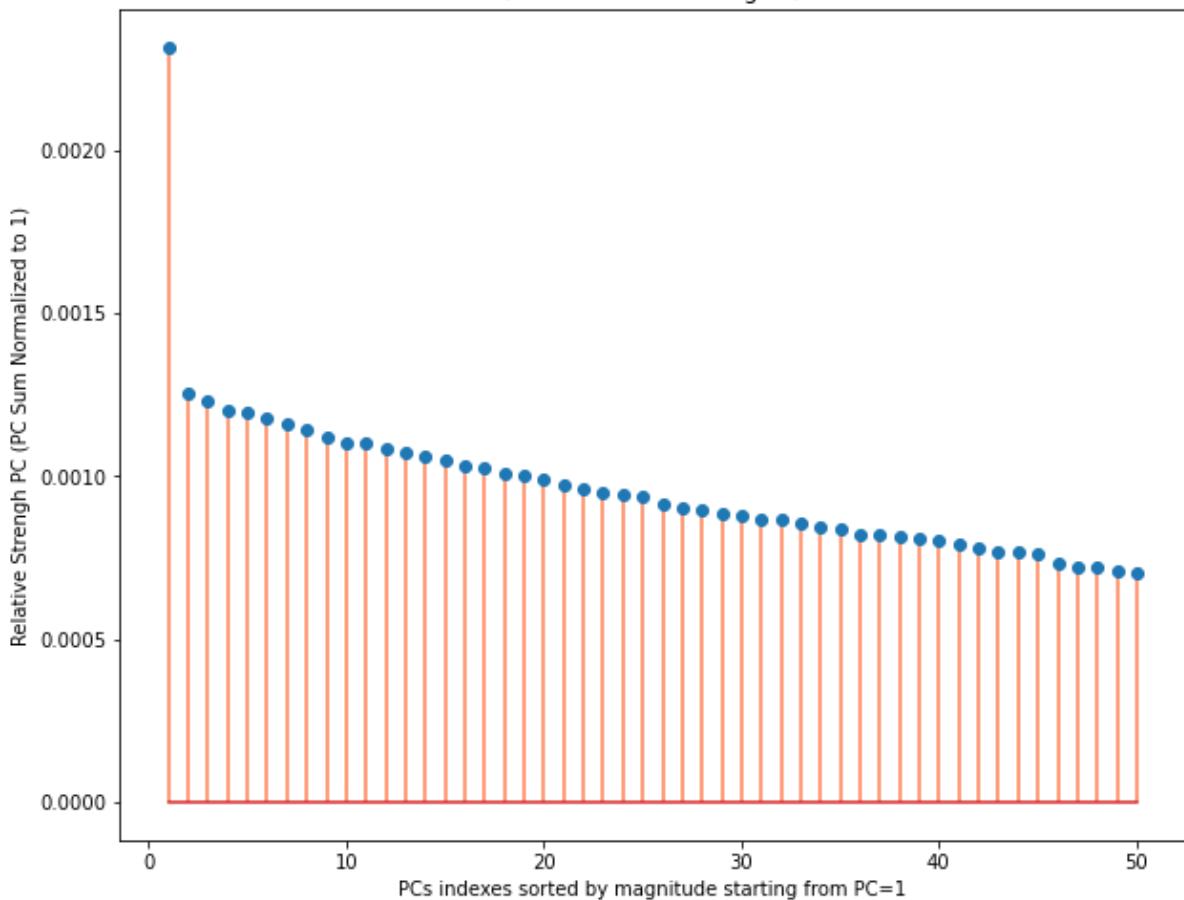


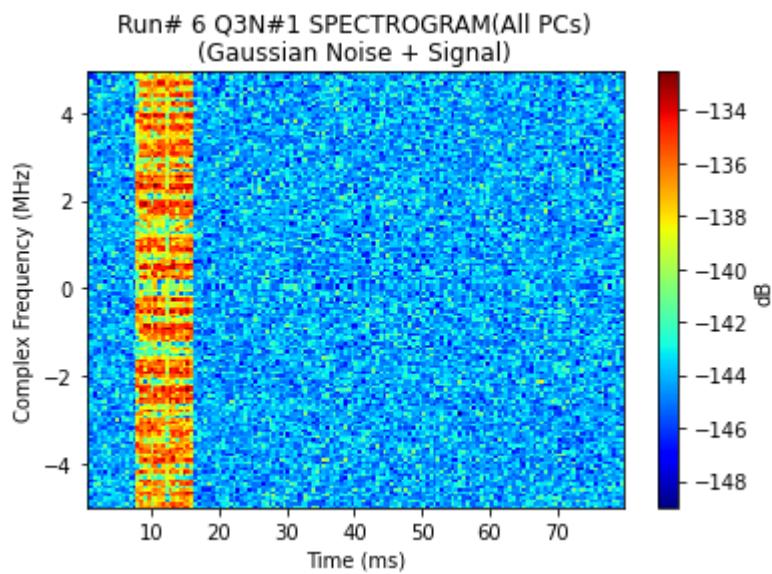
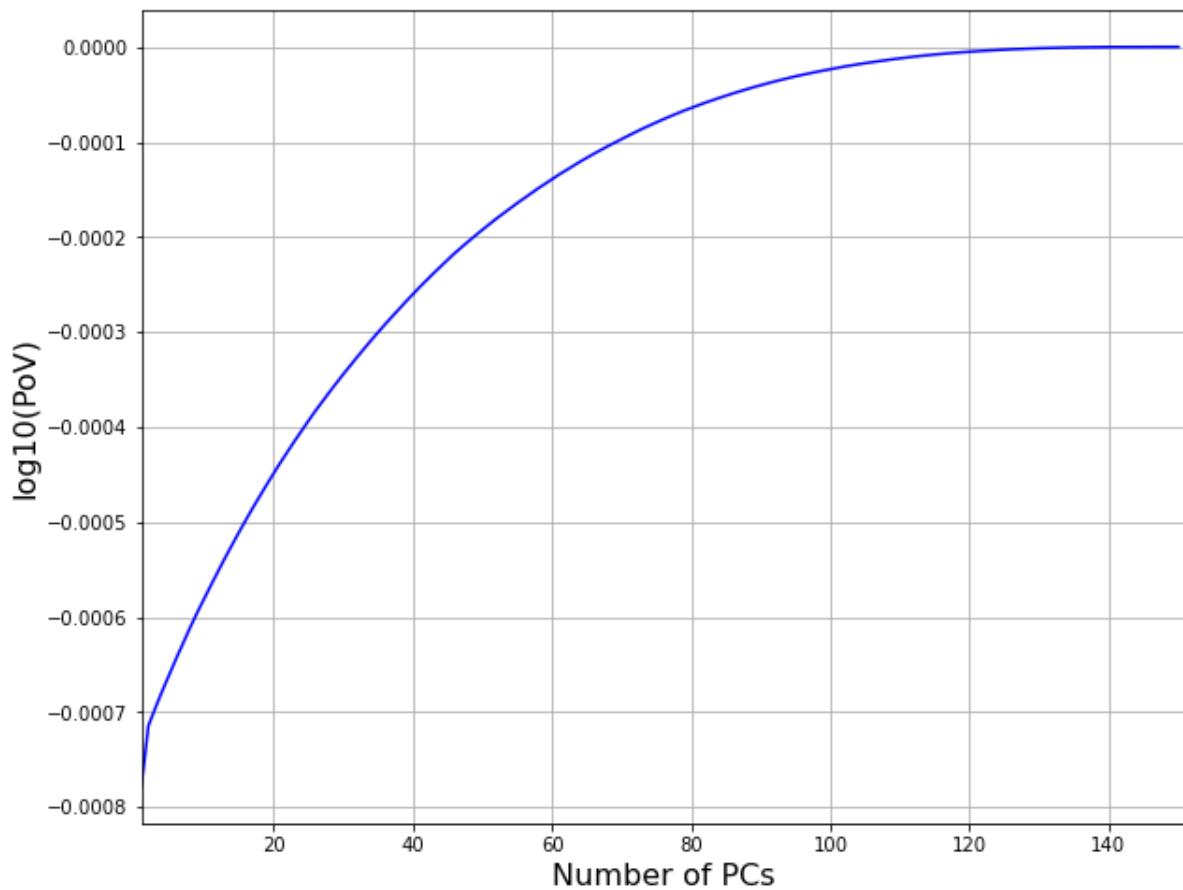


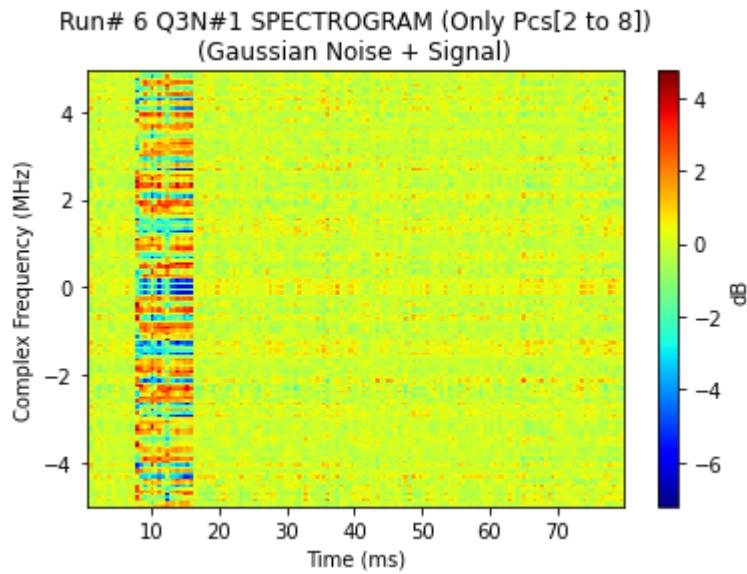
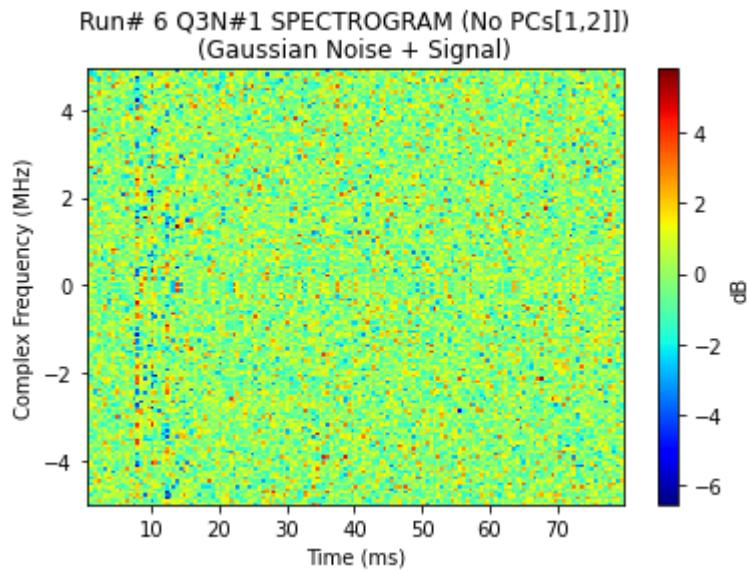
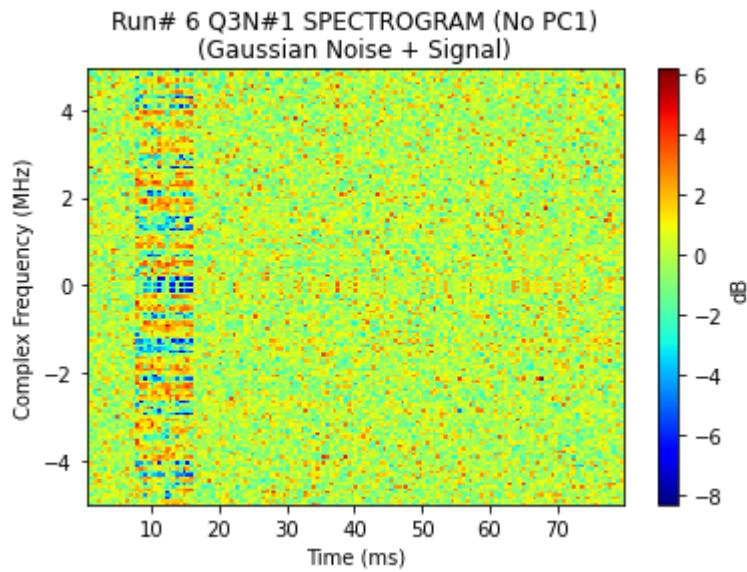
Run# 5 PON#1 SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise + Signal)

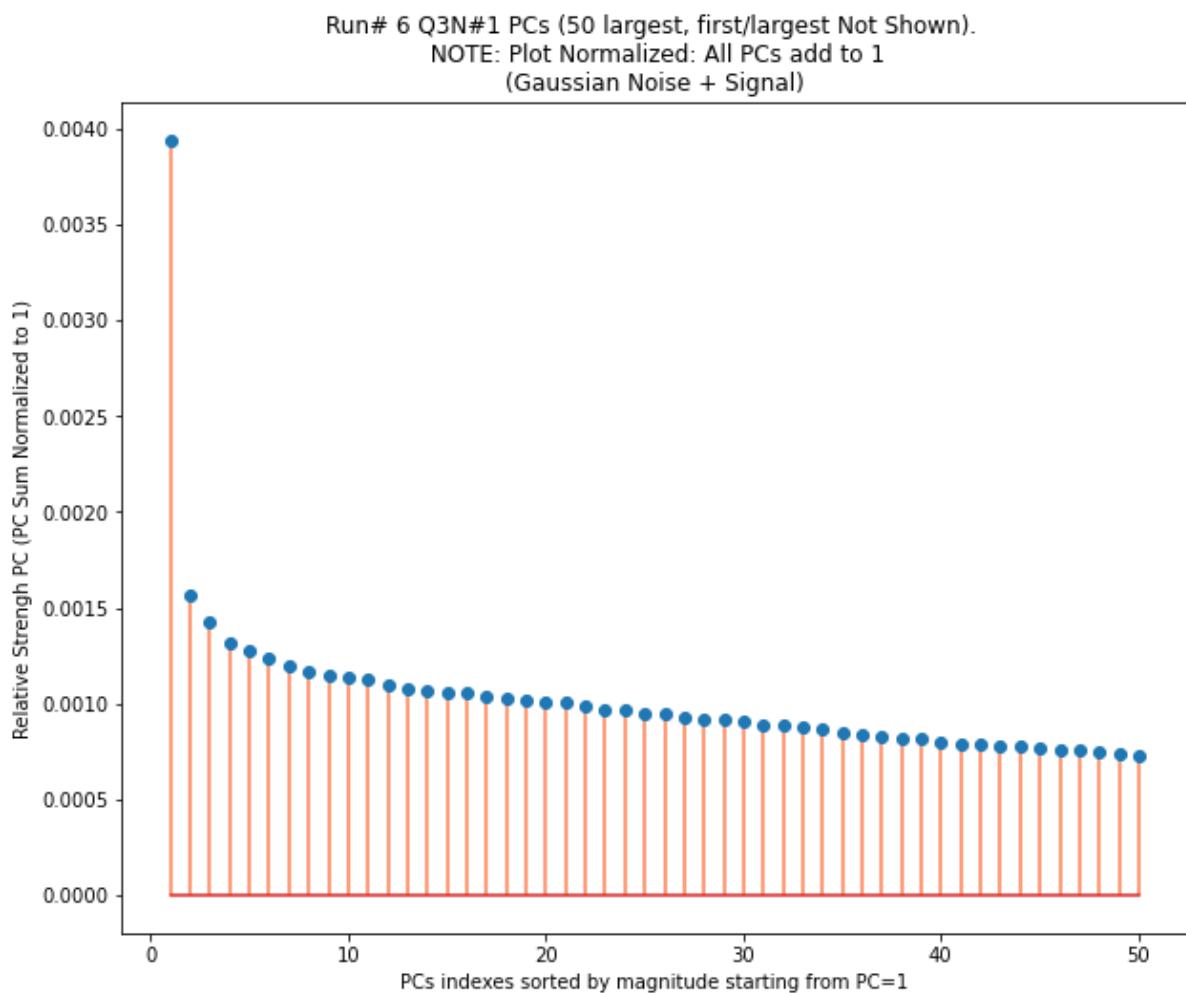
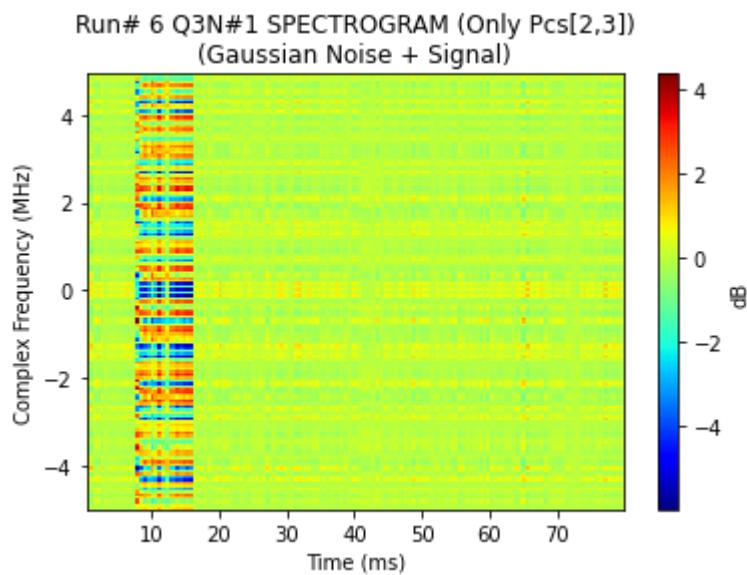


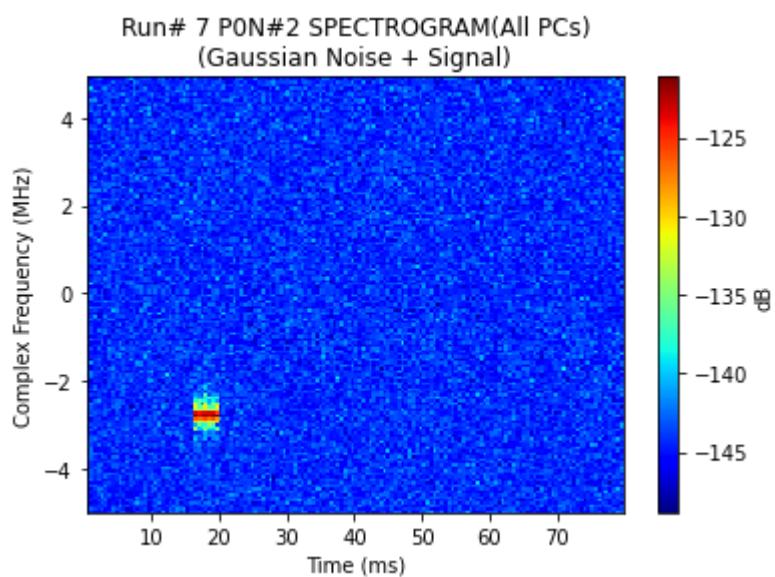
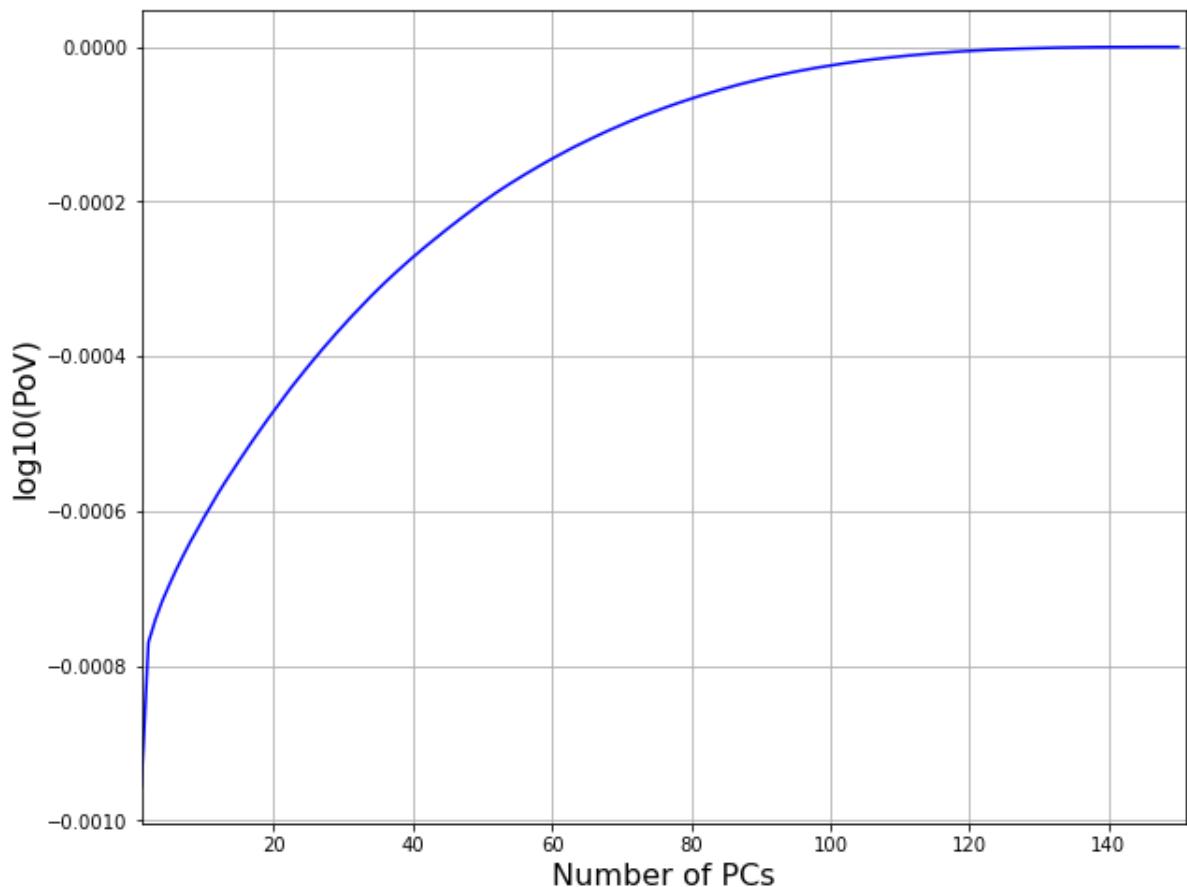
Run# 5 PON#1 PCs (50 largest, first/largest Not Shown).
NOTE: Plot Normalized: All PCs add to 1
(Gaussian Noise + Signal)

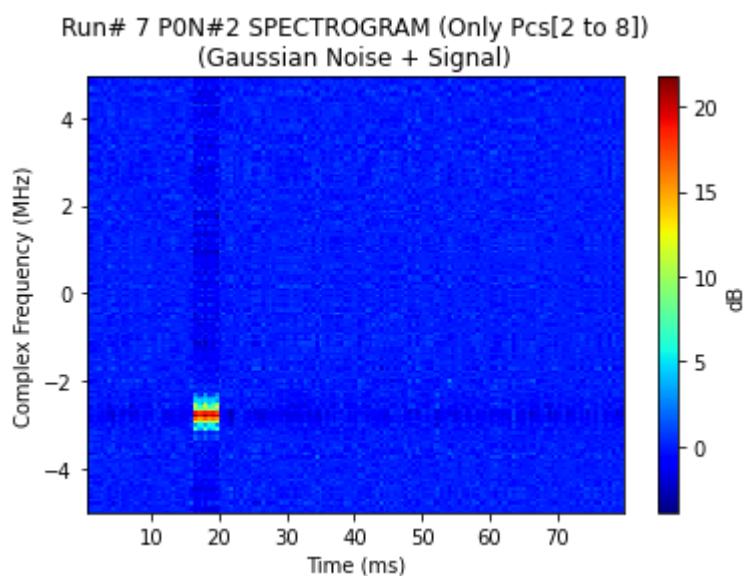
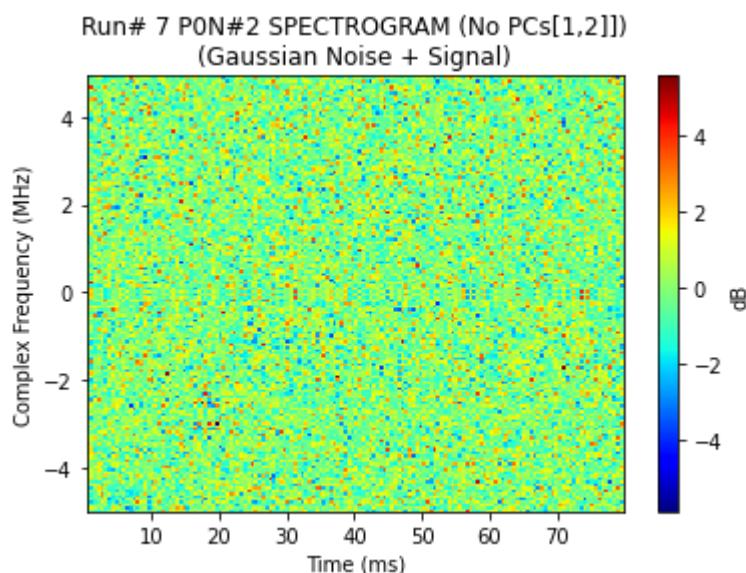
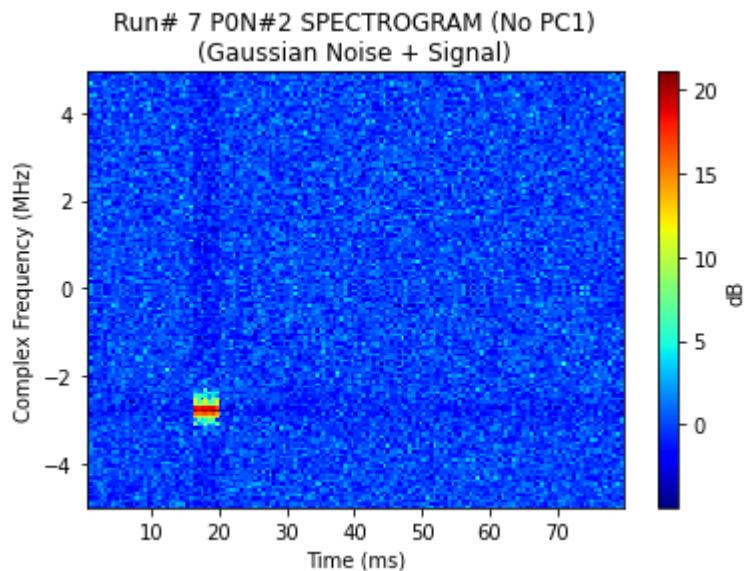


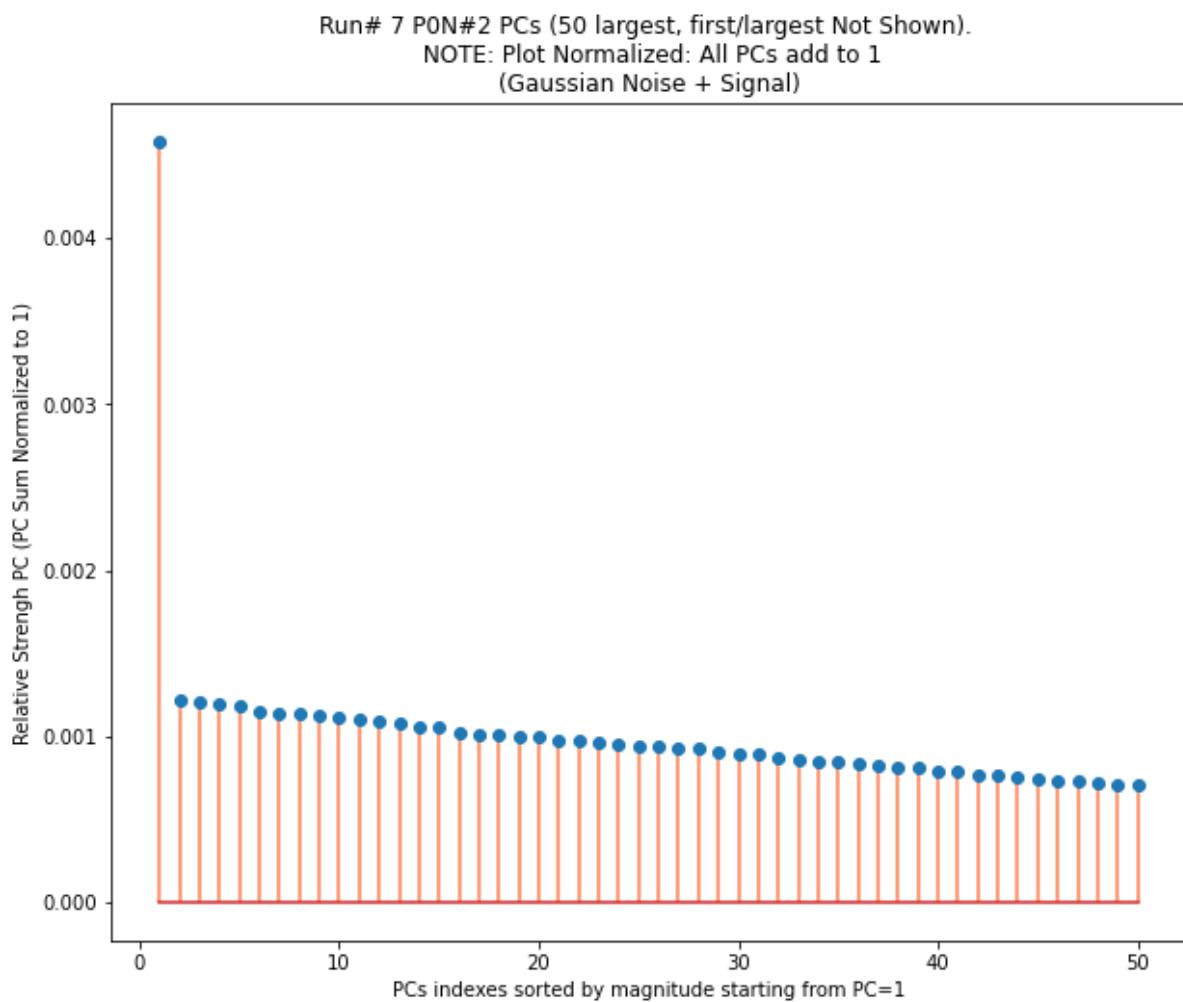
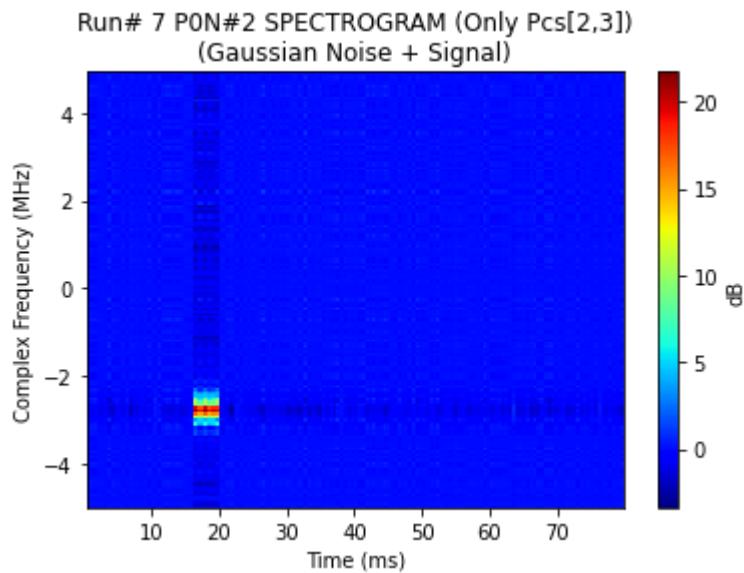


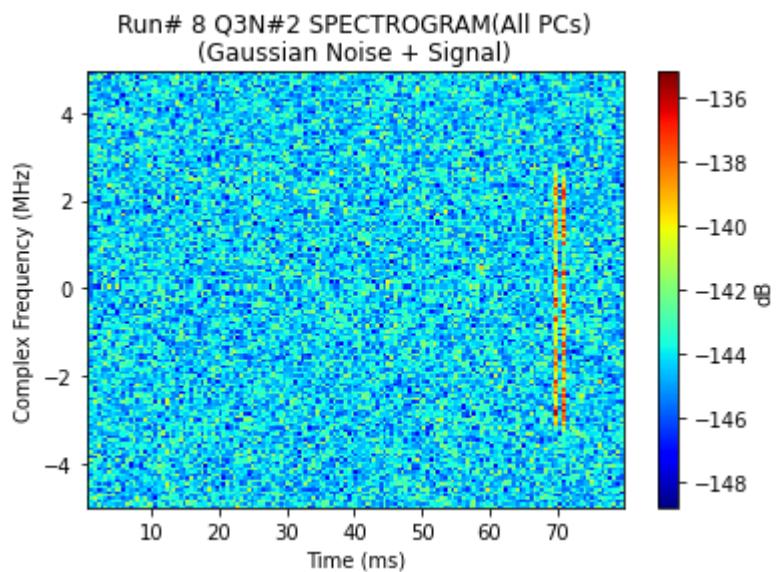
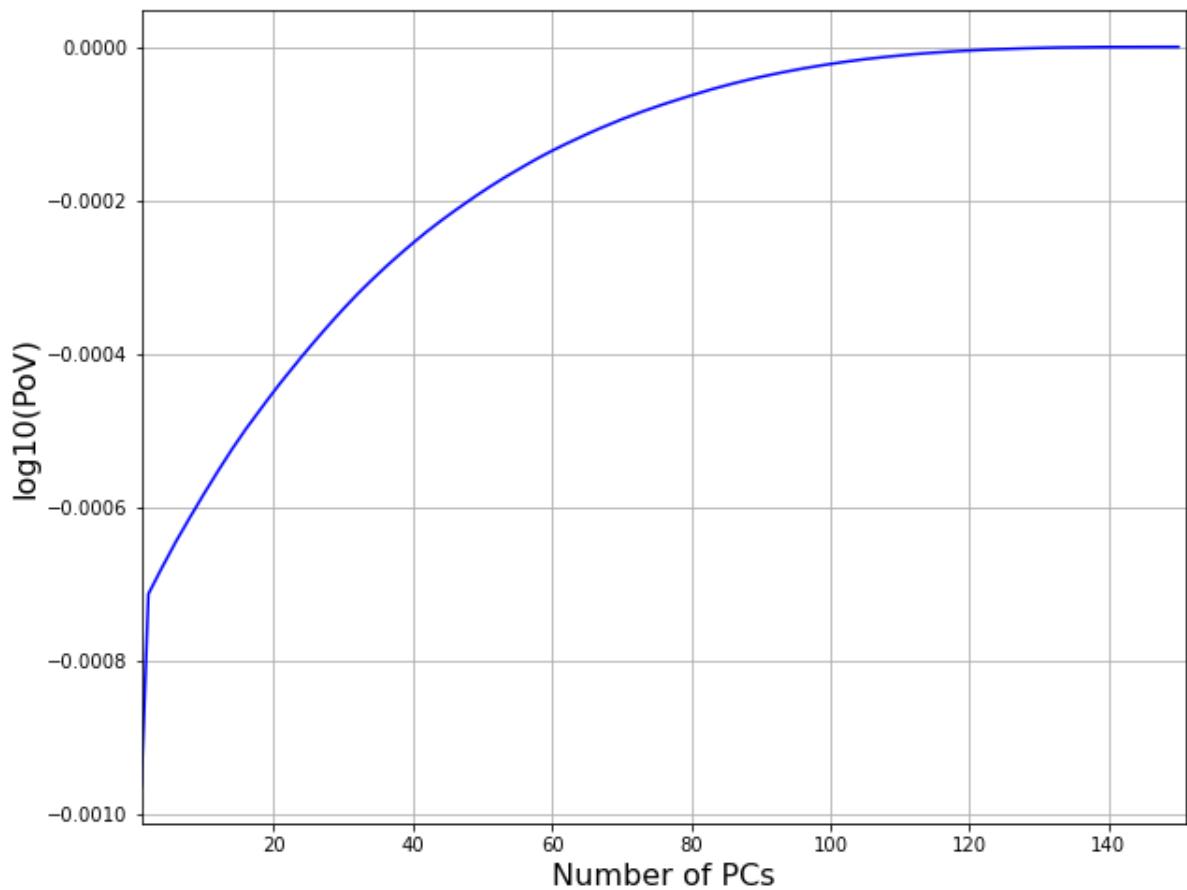


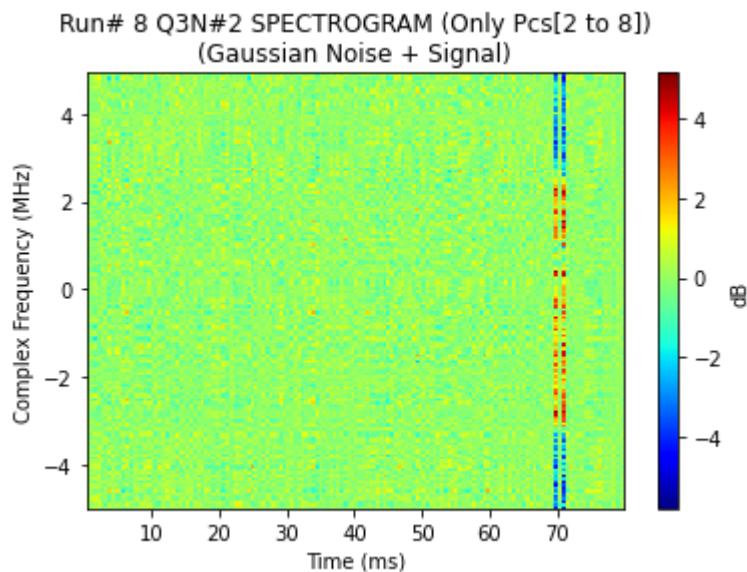
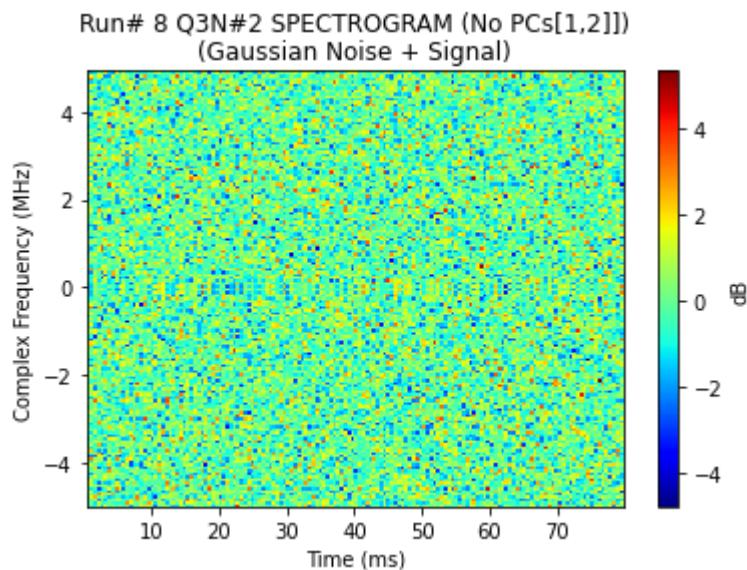
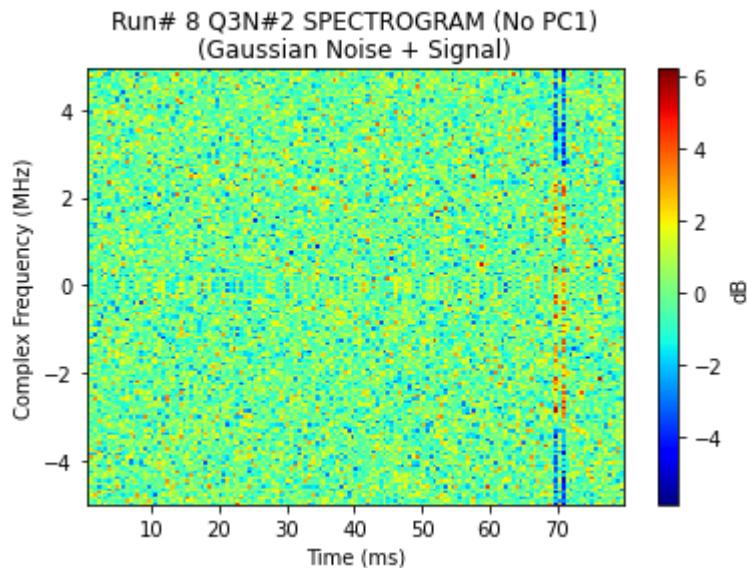


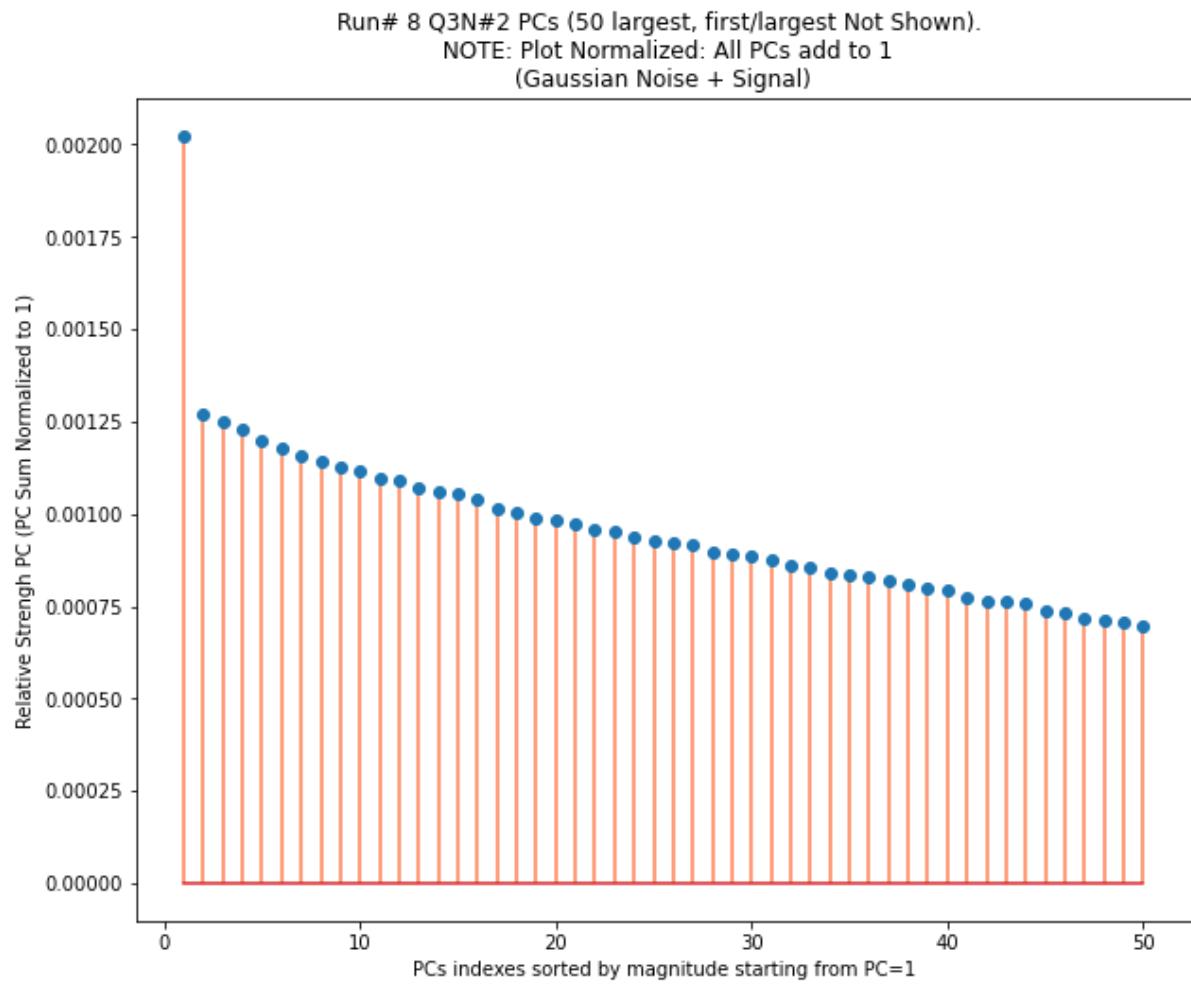
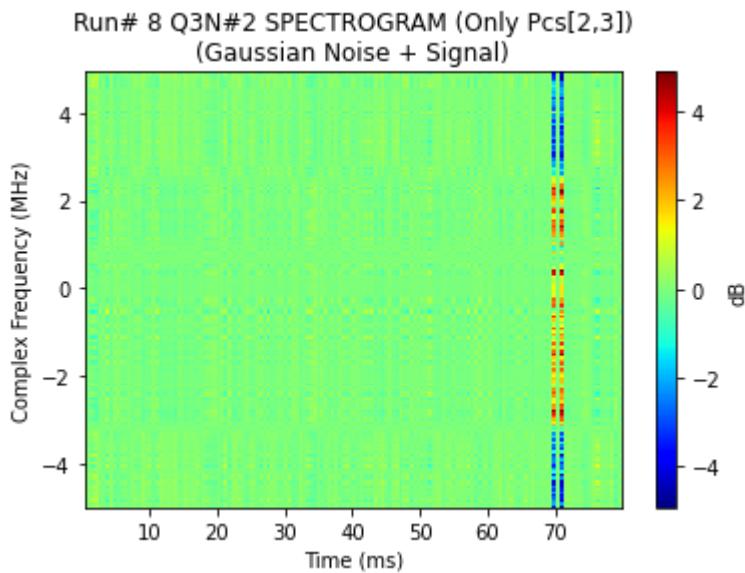


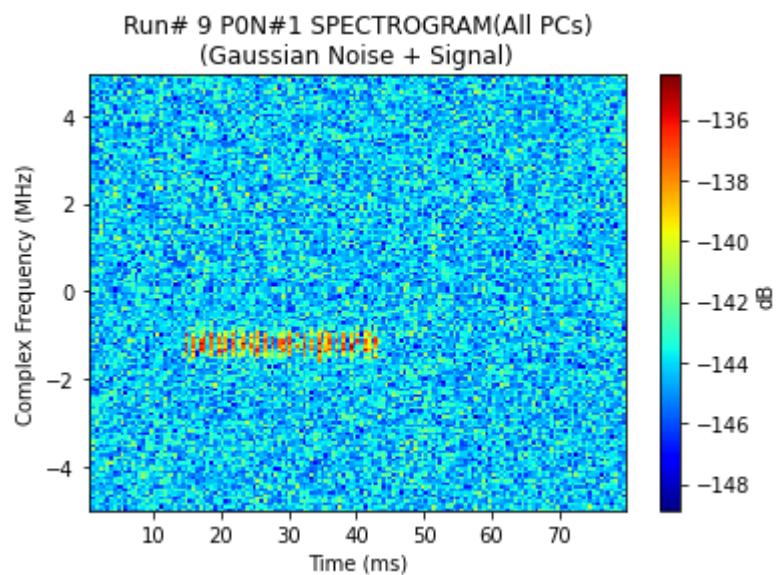
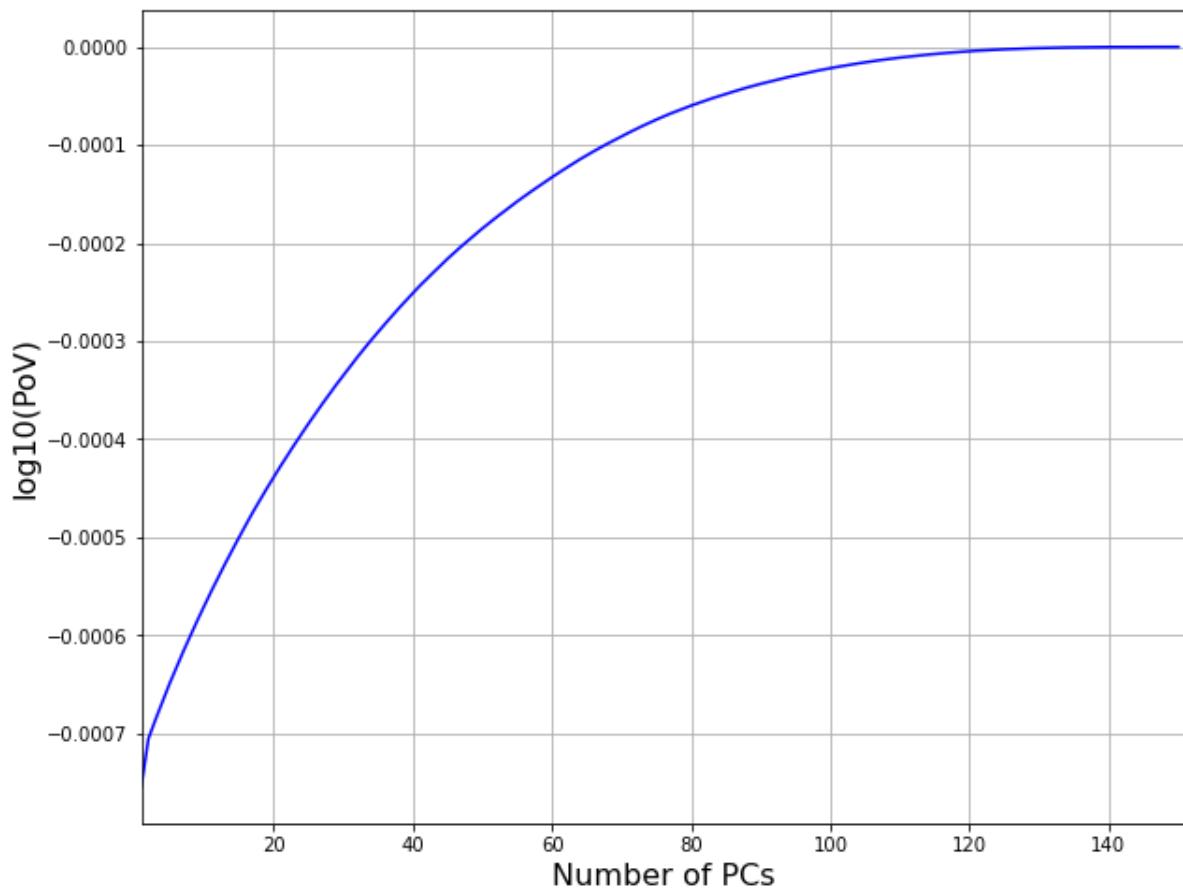


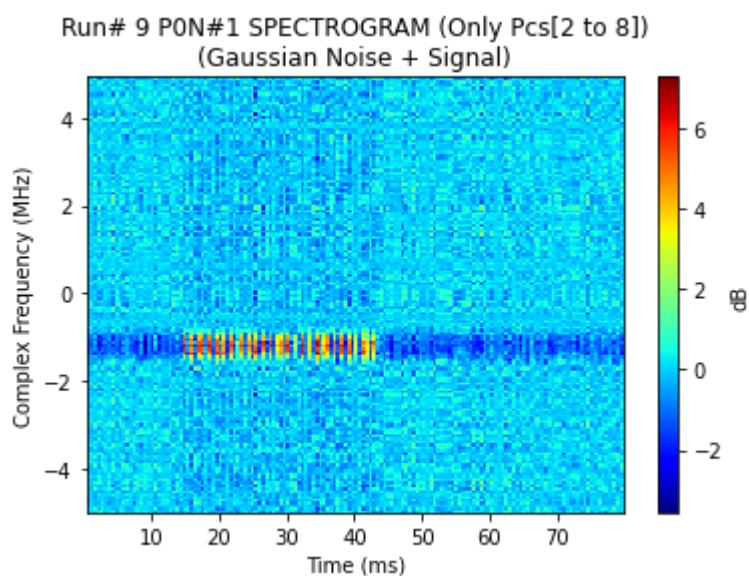
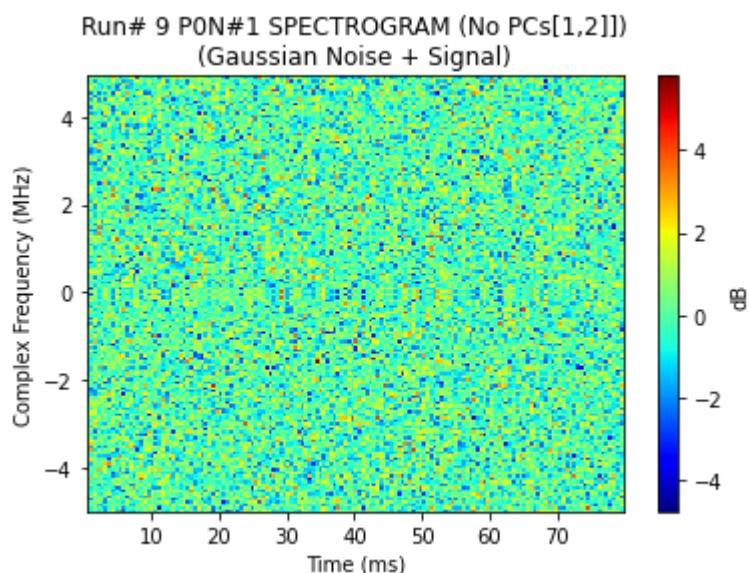
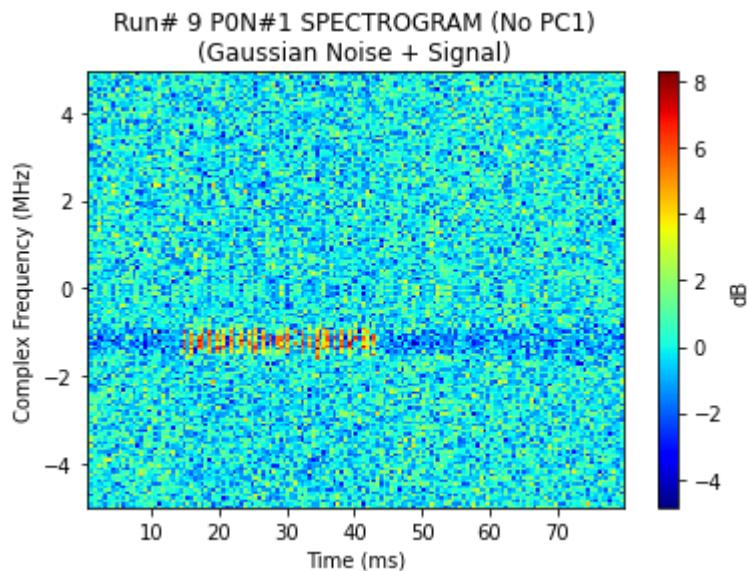




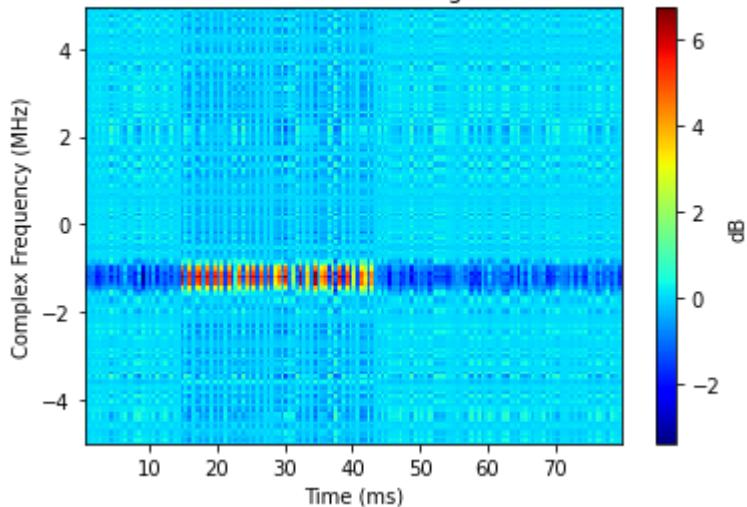




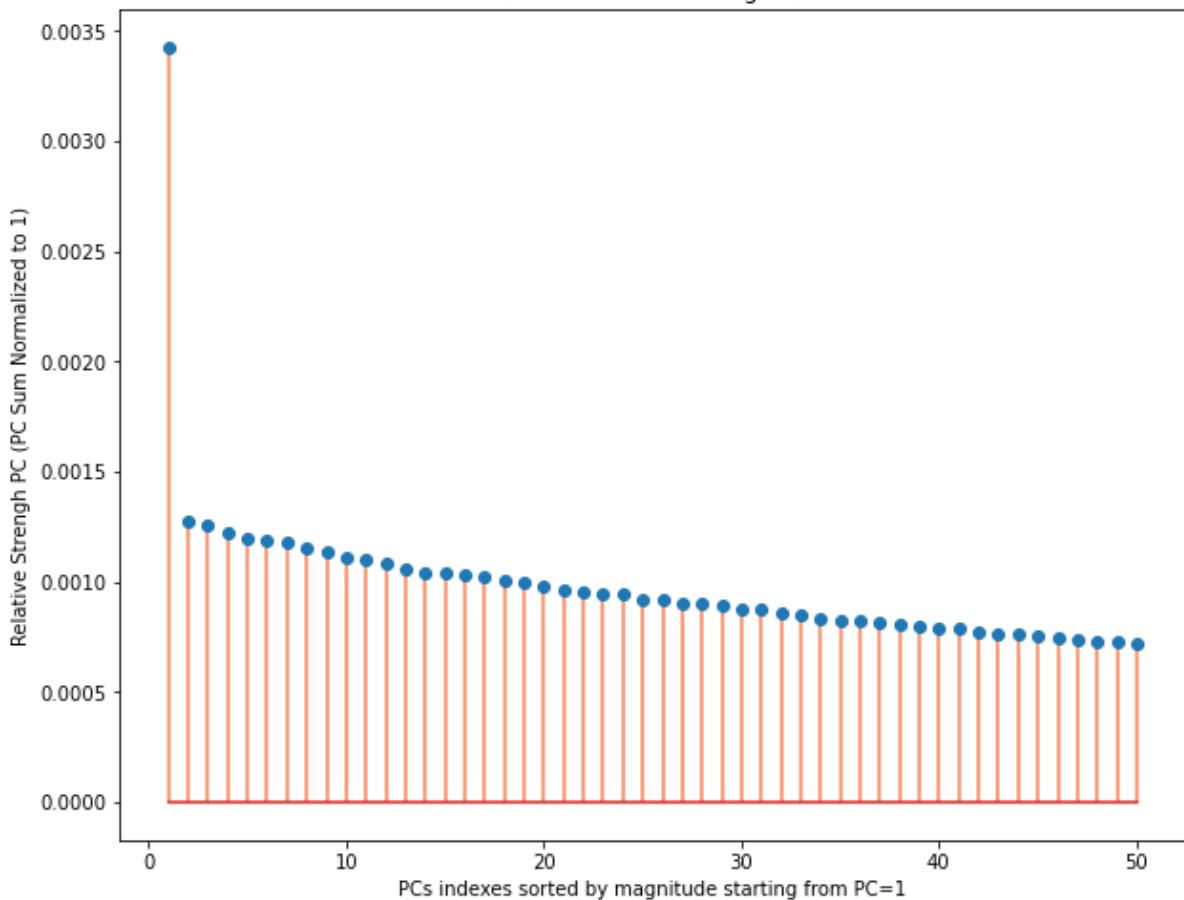


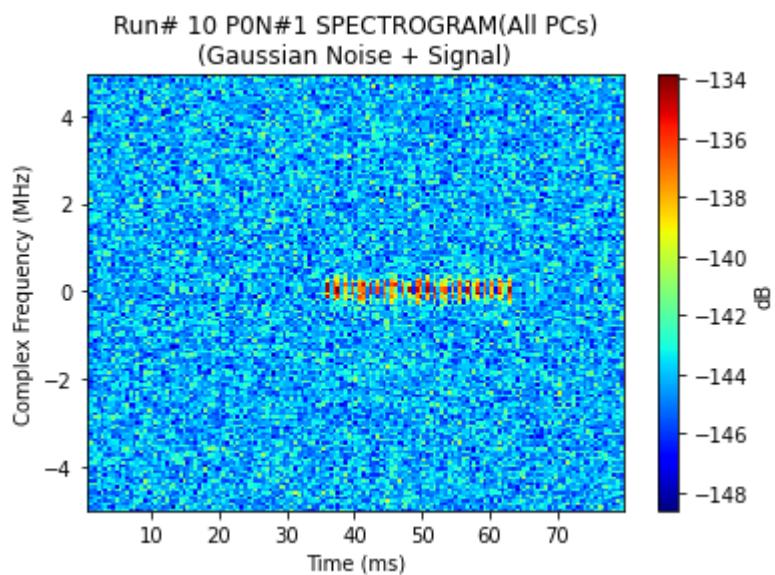
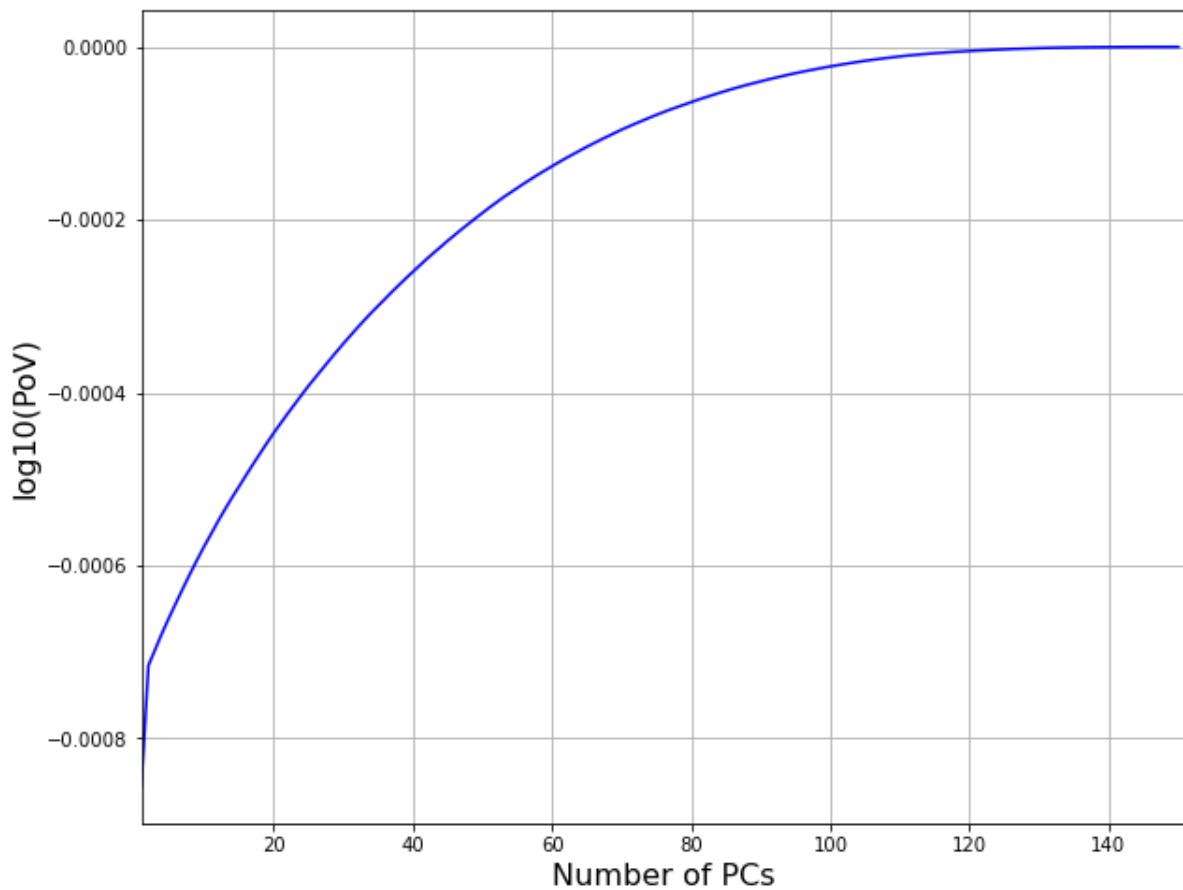


Run# 9 PON#1 SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise + Signal)

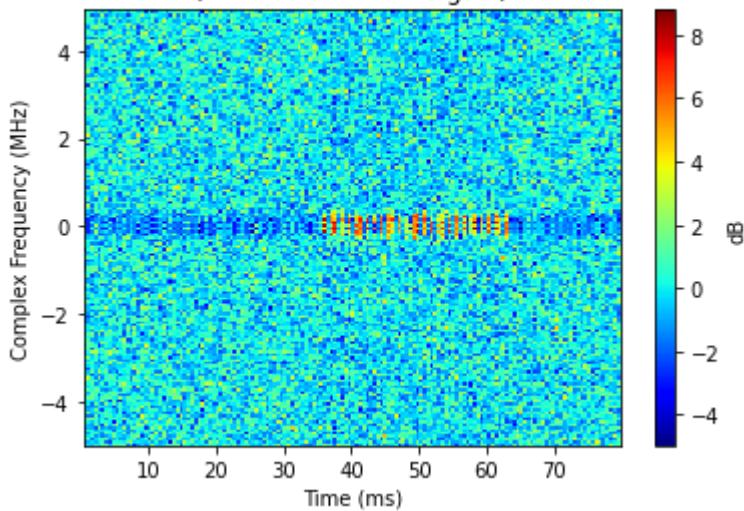


Run# 9 PON#1 PCs (50 largest, first/largest Not Shown).
NOTE: Plot Normalized: All PCs add to 1
(Gaussian Noise + Signal)

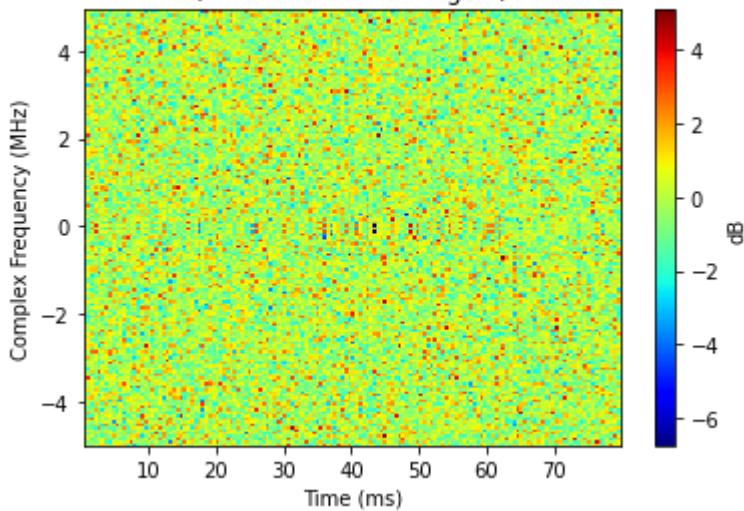




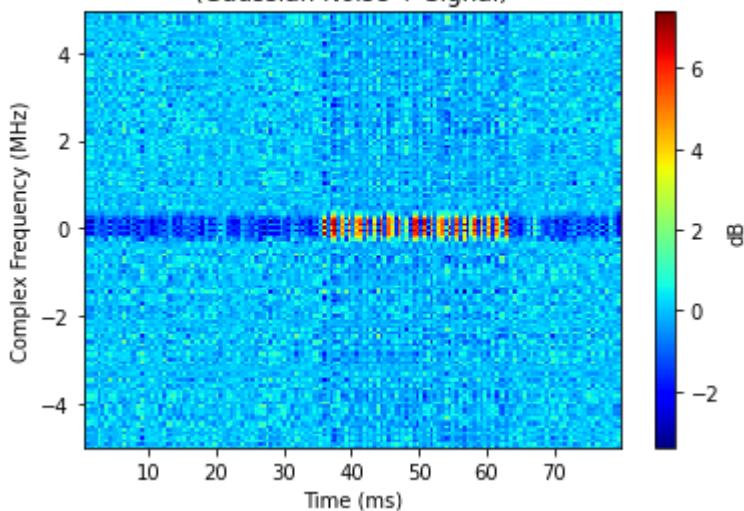
Run# 10 PON#1 SPECTROGRAM (No PC1)
(Gaussian Noise + Signal)



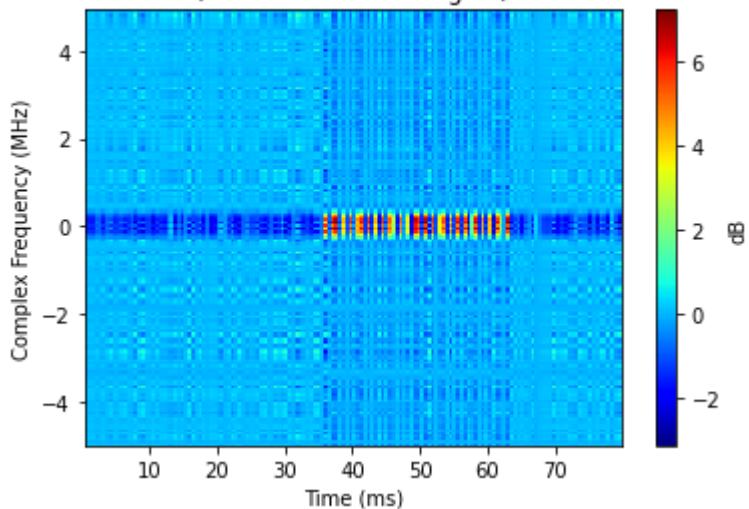
Run# 10 PON#1 SPECTROGRAM (No PCs[1,2])
(Gaussian Noise + Signal)



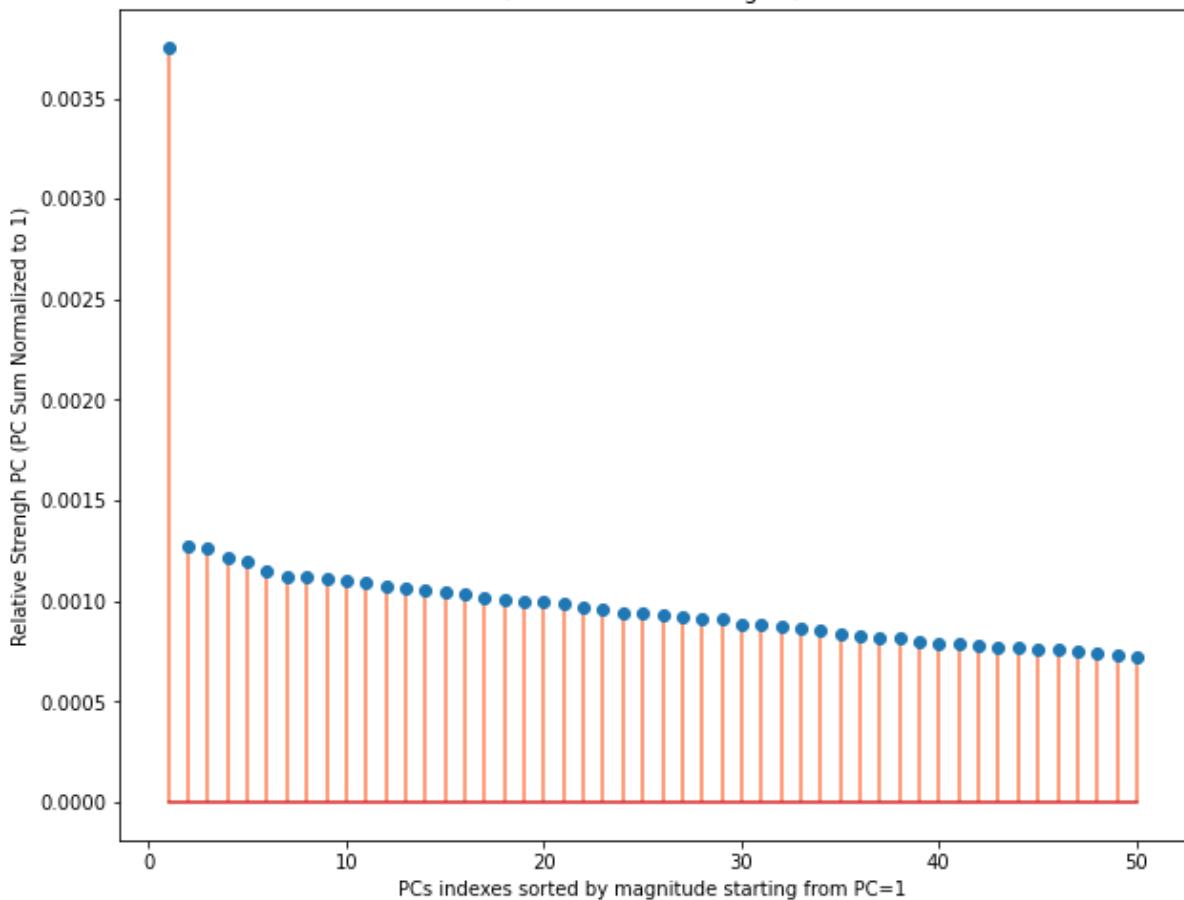
Run# 10 PON#1 SPECTROGRAM (Only Pcs[2 to 8])
(Gaussian Noise + Signal)

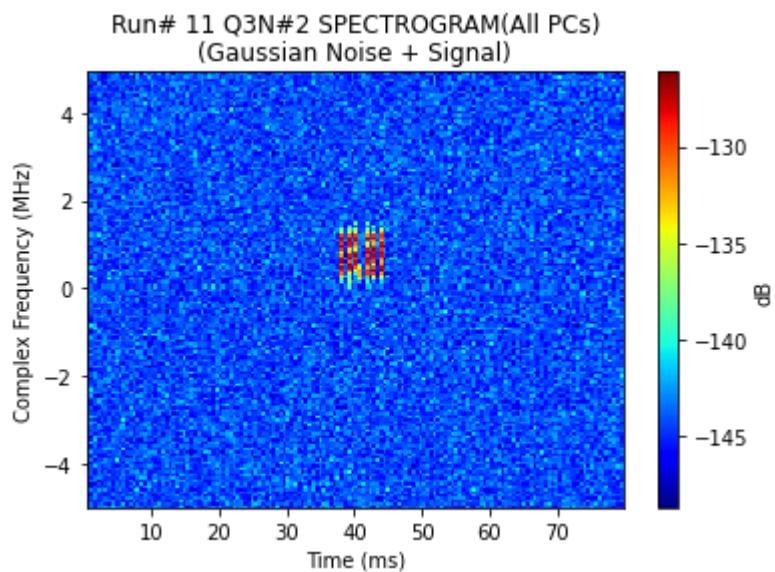
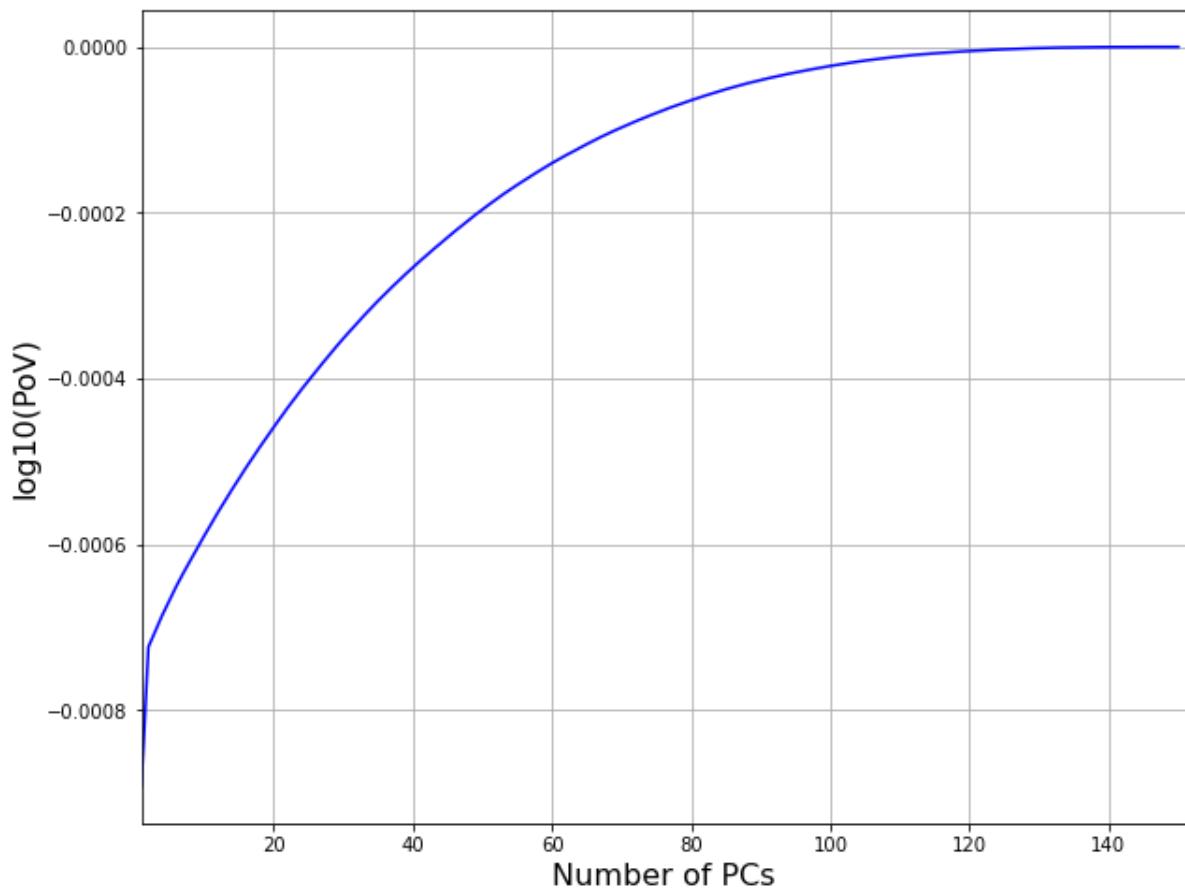


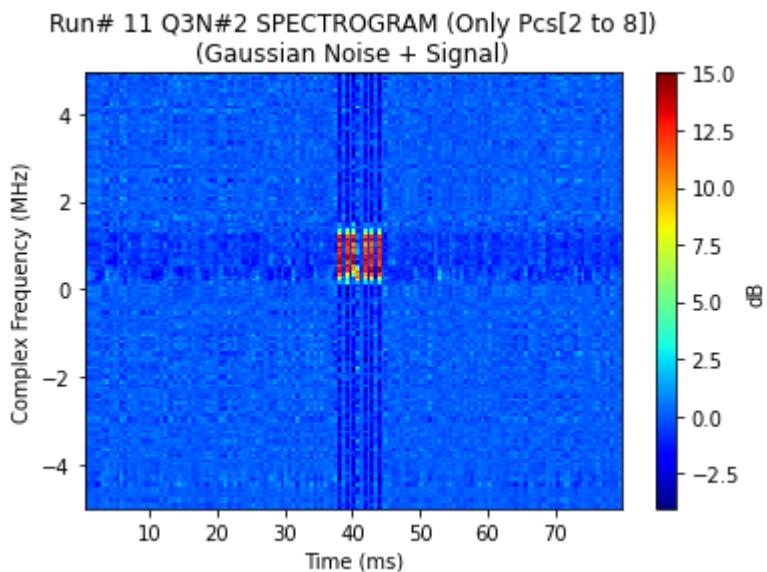
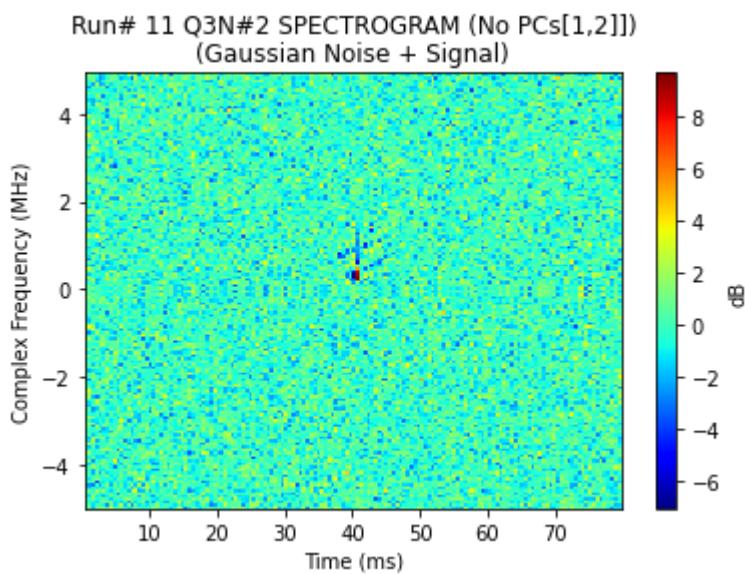
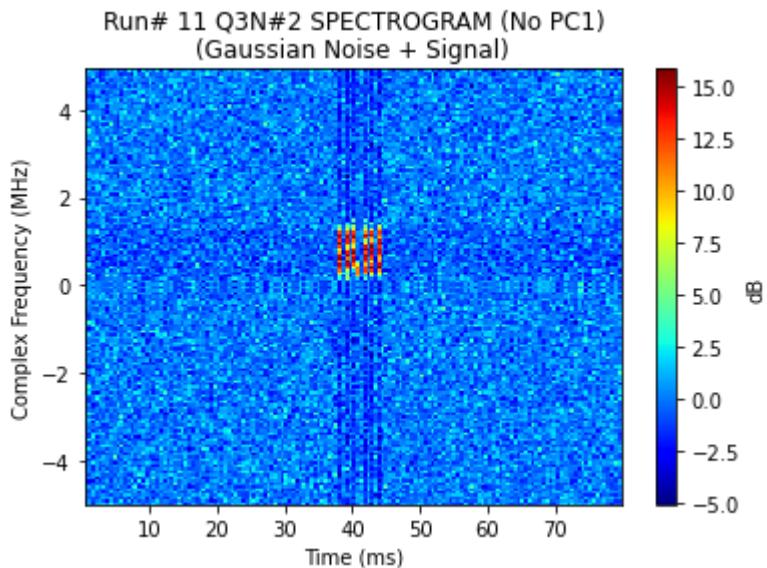
Run# 10 PON#1 SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise + Signal)



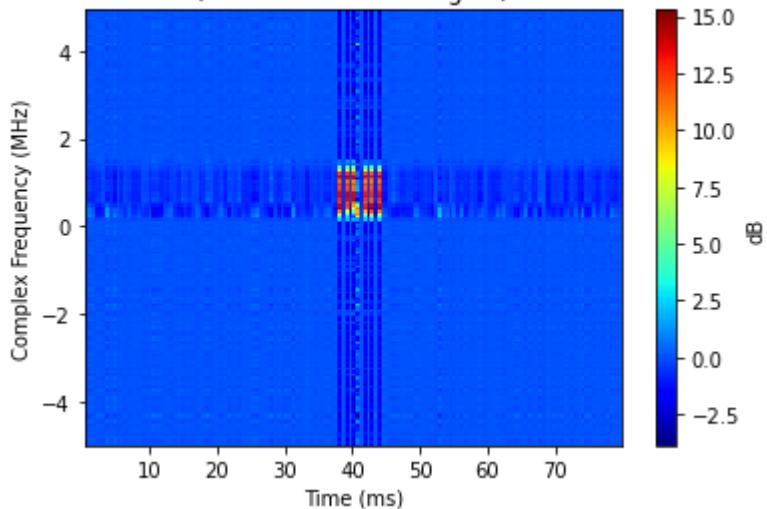
Run# 10 PON#1 PCs (50 largest, first/largest Not Shown).
NOTE: Plot Normalized: All PCs add to 1
(Gaussian Noise + Signal)



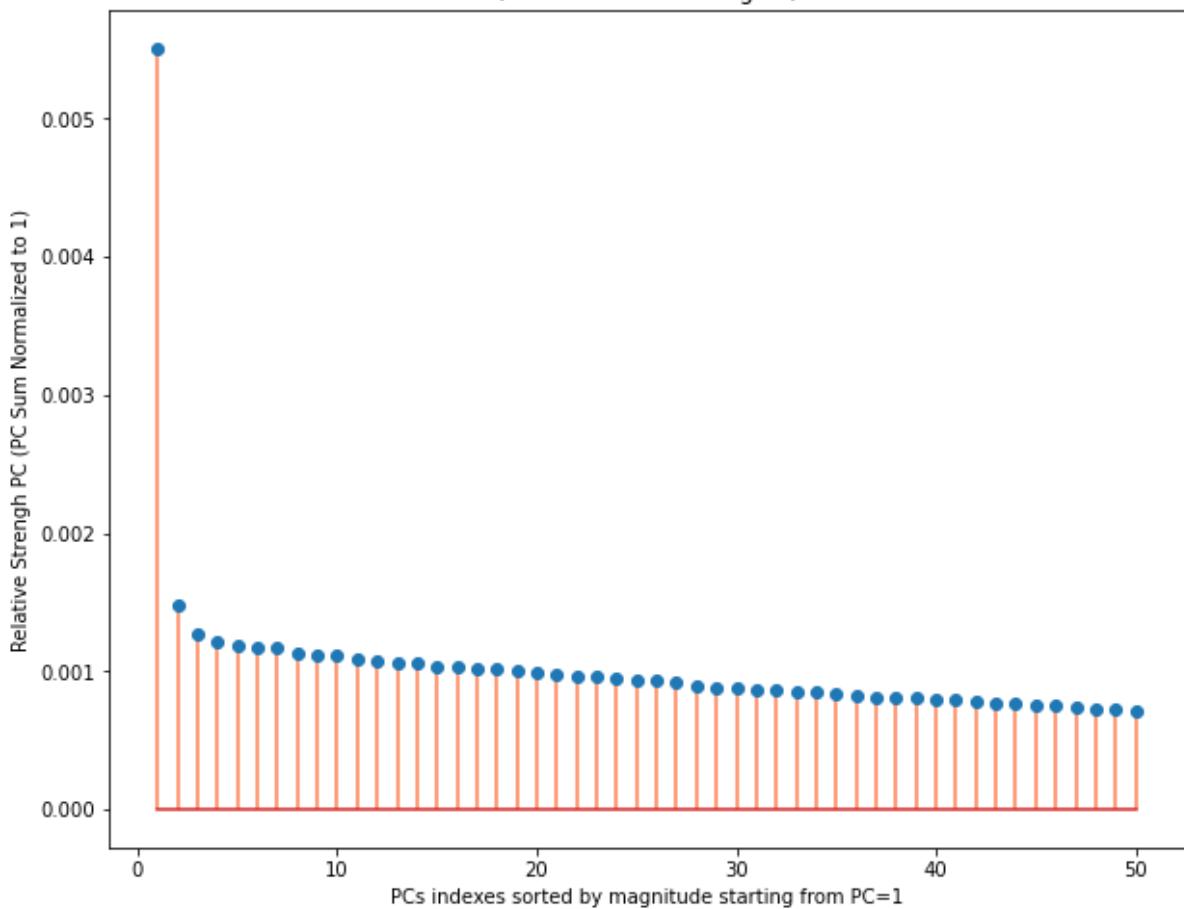


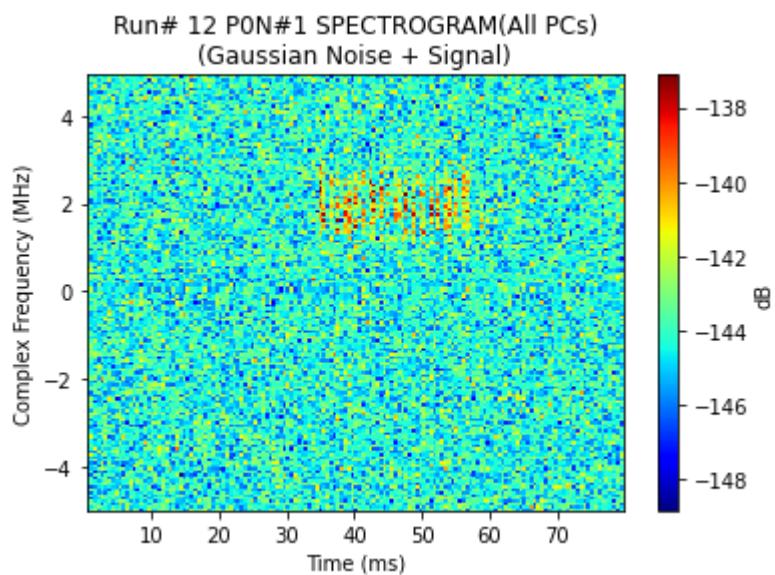
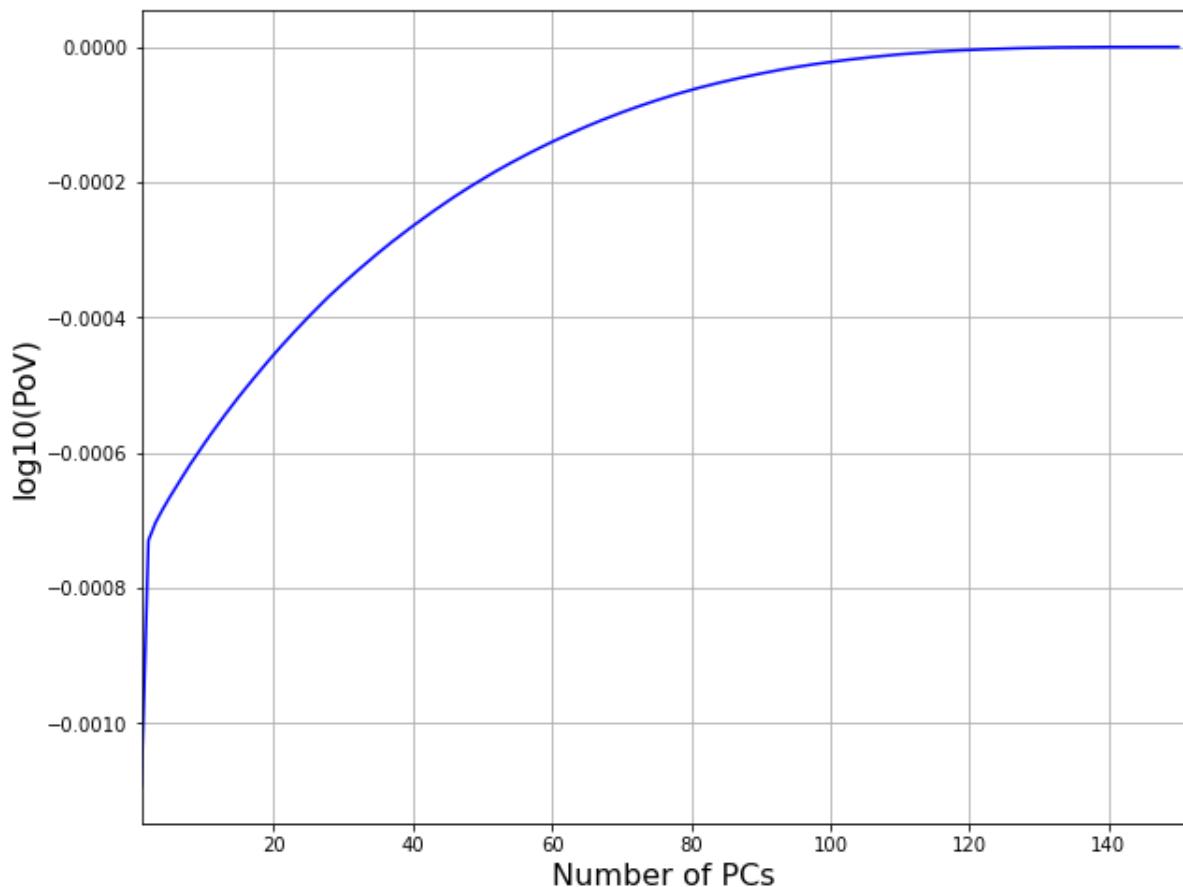


Run# 11 Q3N#2 SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise + Signal)

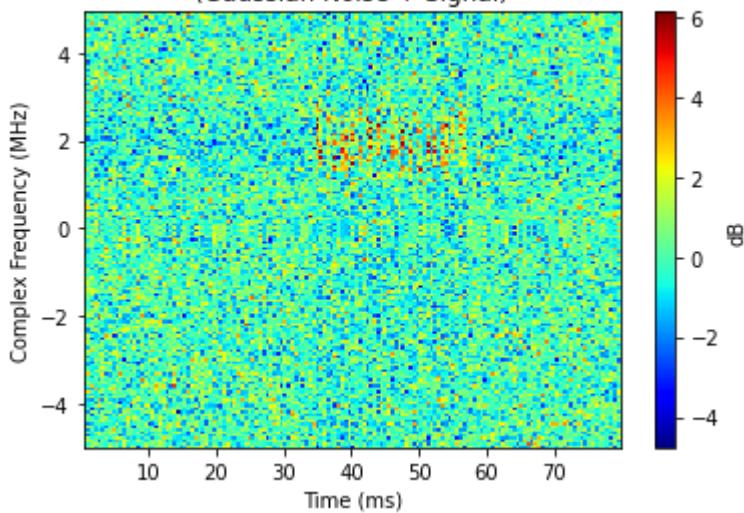


Run# 11 Q3N#2 PCs (50 largest, first/largest Not Shown).
NOTE: Plot Normalized: All PCs add to 1
(Gaussian Noise + Signal)

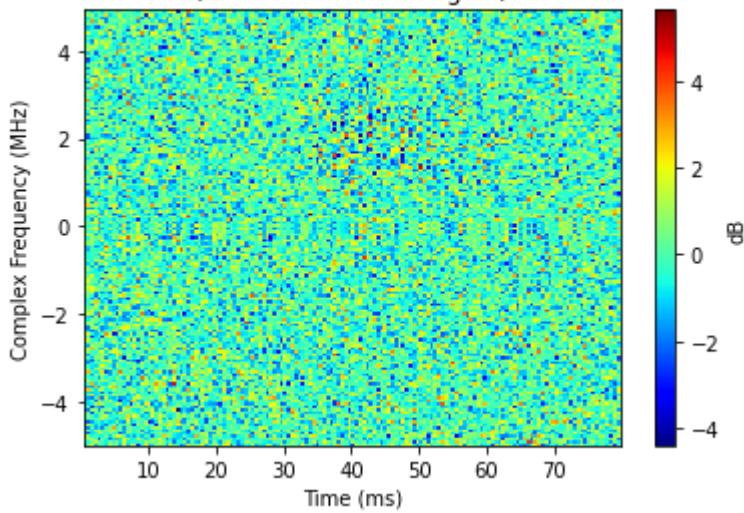




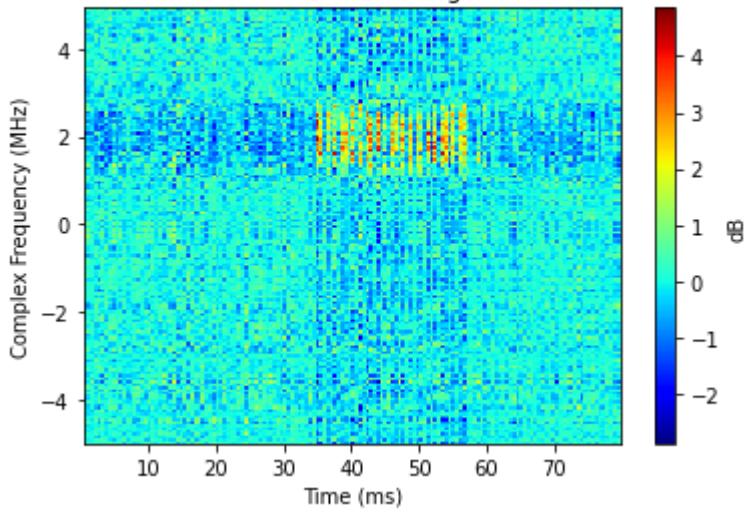
Run# 12 PON#1 SPECTROGRAM (No PC1)
(Gaussian Noise + Signal)



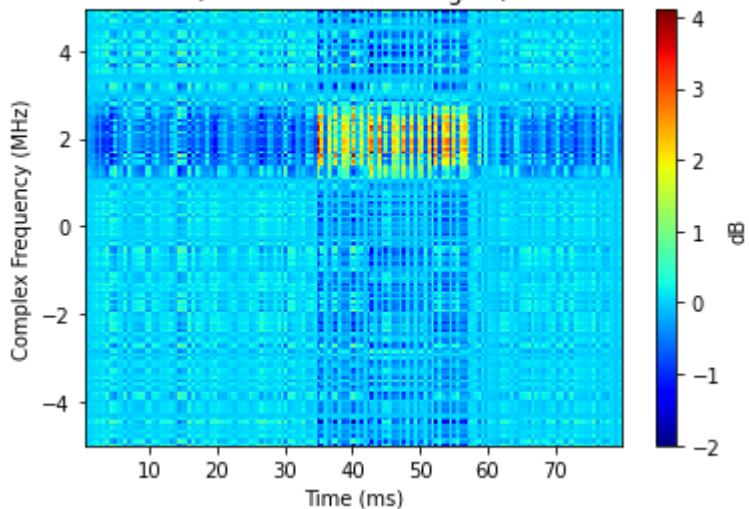
Run# 12 PON#1 SPECTROGRAM (No PCs[1,2])
(Gaussian Noise + Signal)



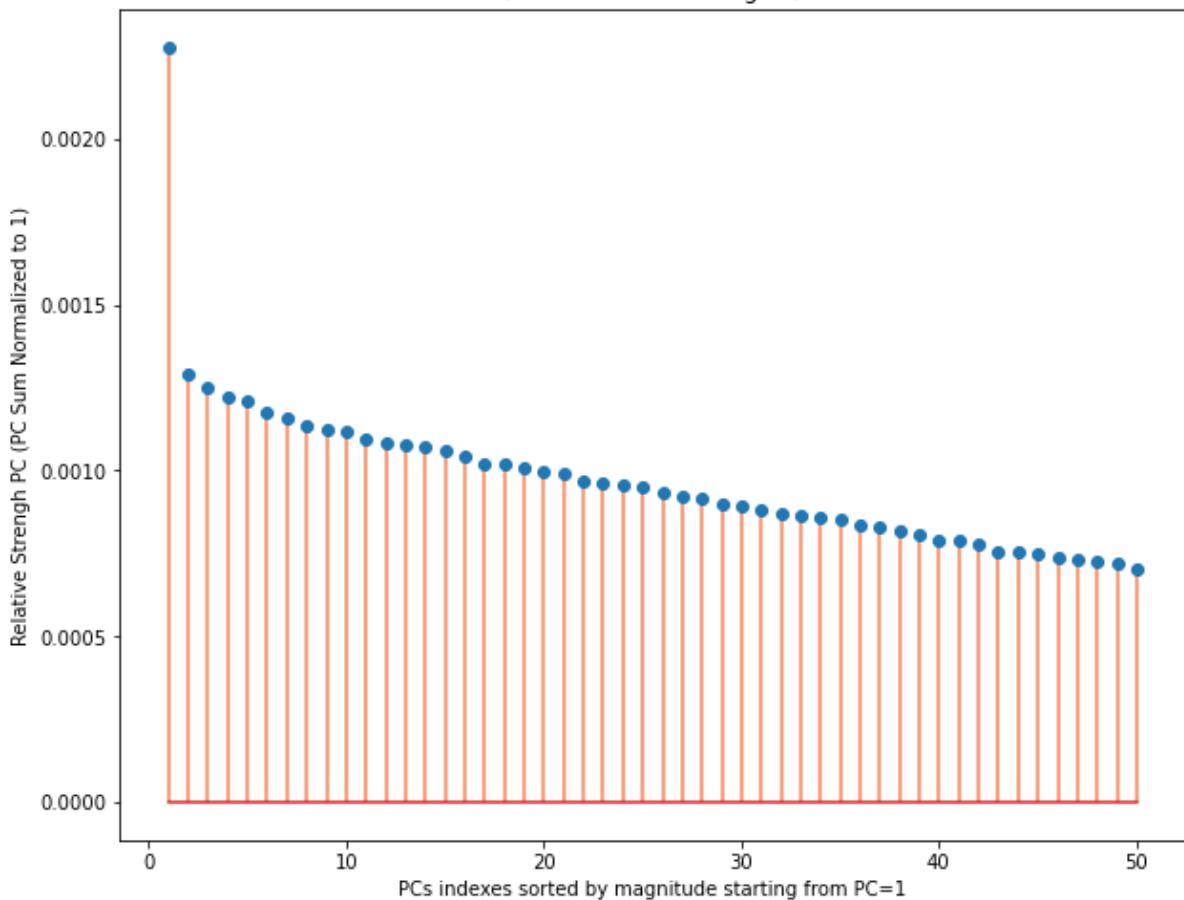
Run# 12 PON#1 SPECTROGRAM (Only Pcs[2 to 8])
(Gaussian Noise + Signal)

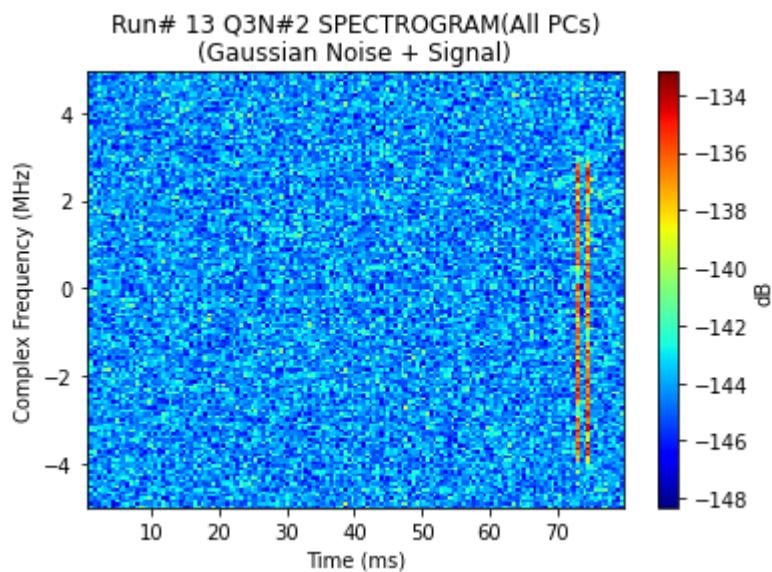
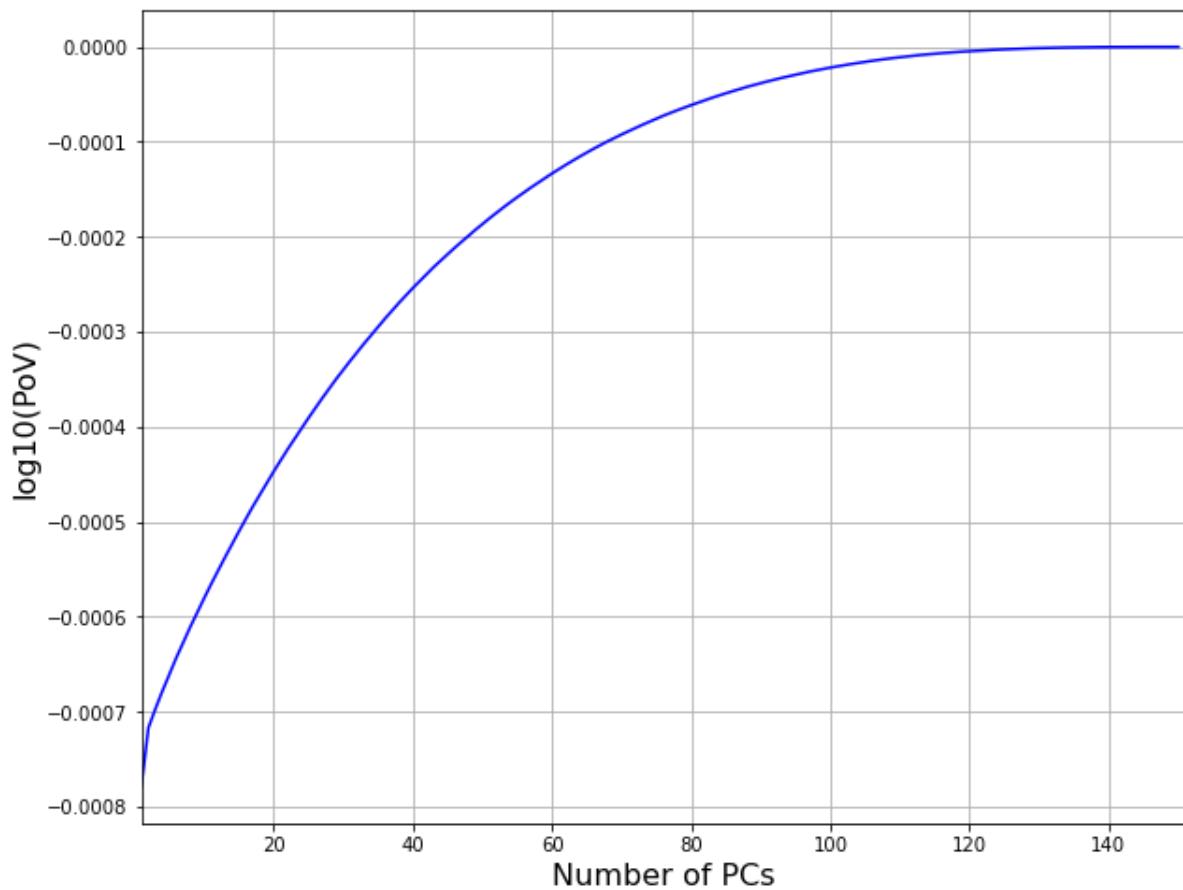


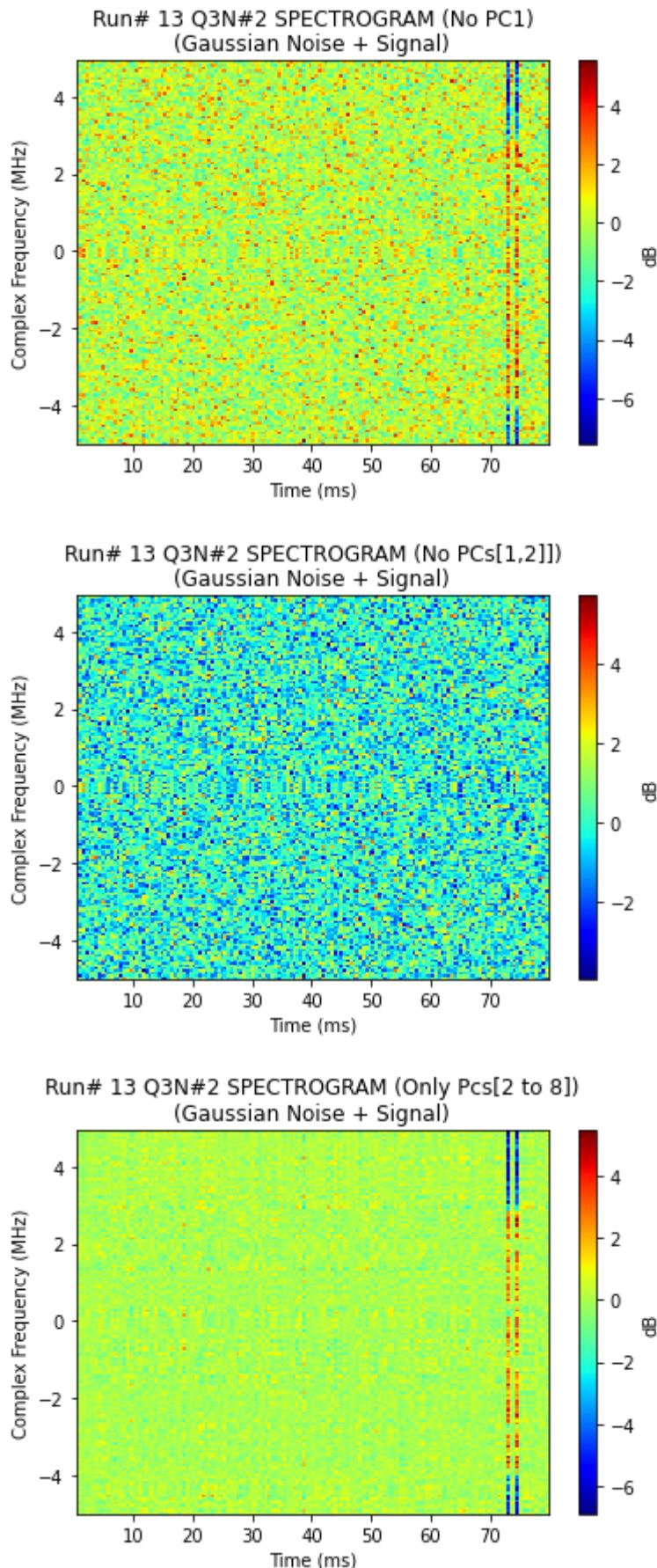
Run# 12 PON#1 SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise + Signal)

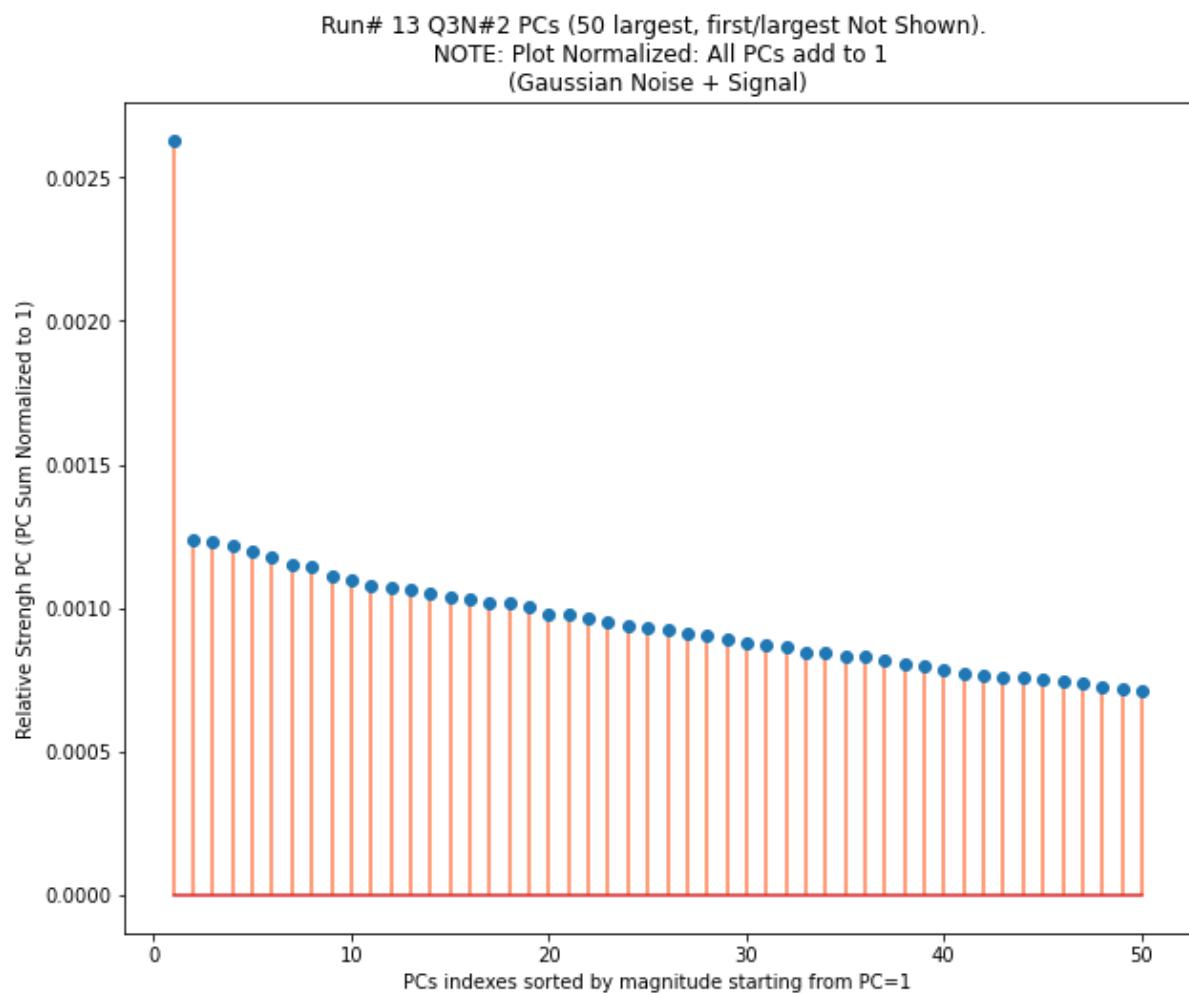
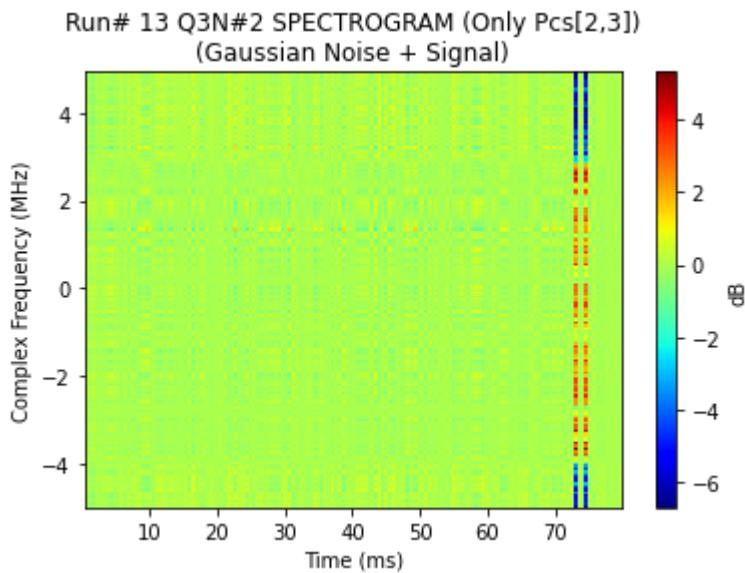


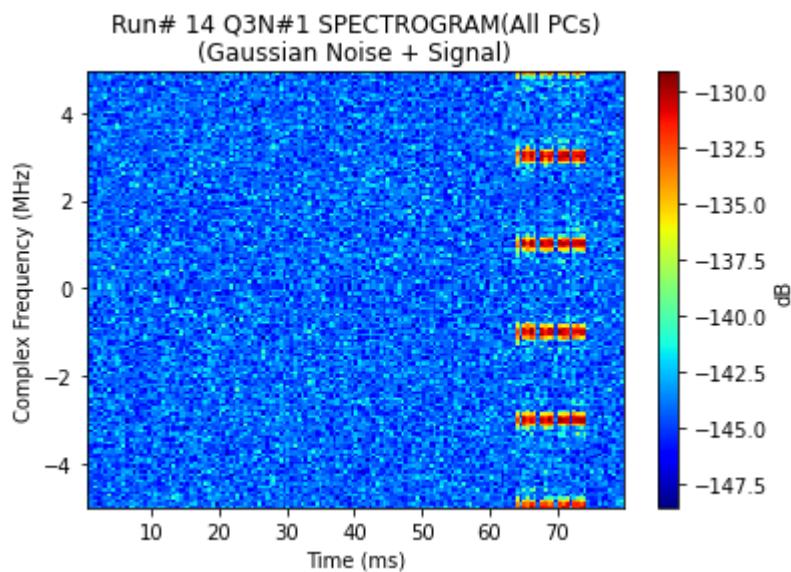
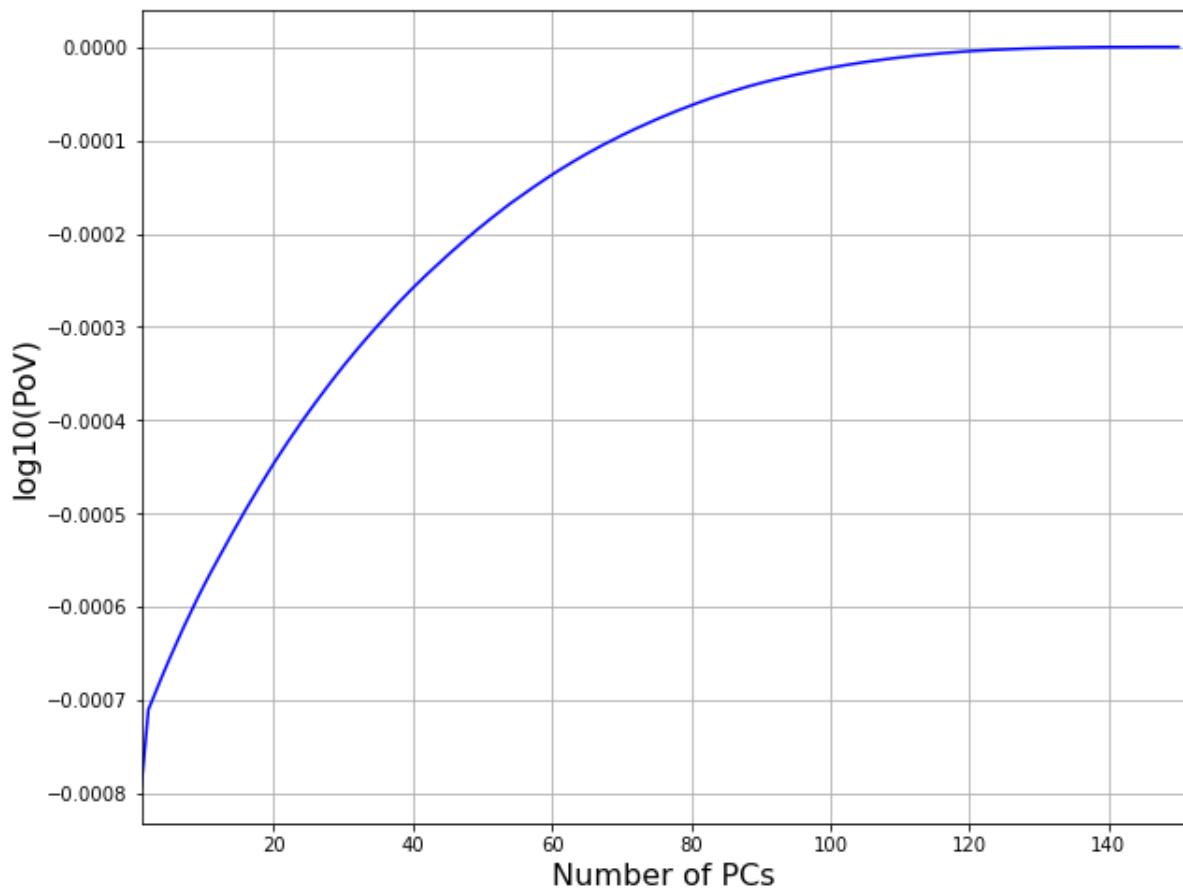
Run# 12 PON#1 PCs (50 largest, first/largest Not Shown).
NOTE: Plot Normalized: All PCs add to 1
(Gaussian Noise + Signal)

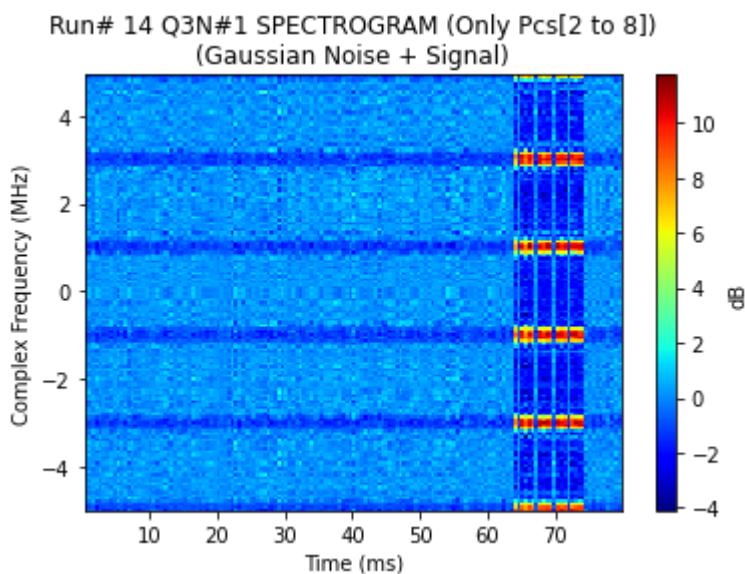
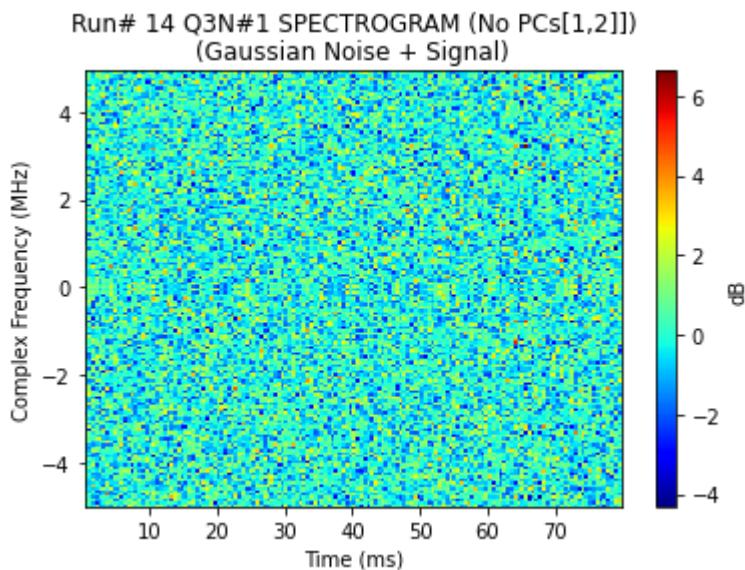
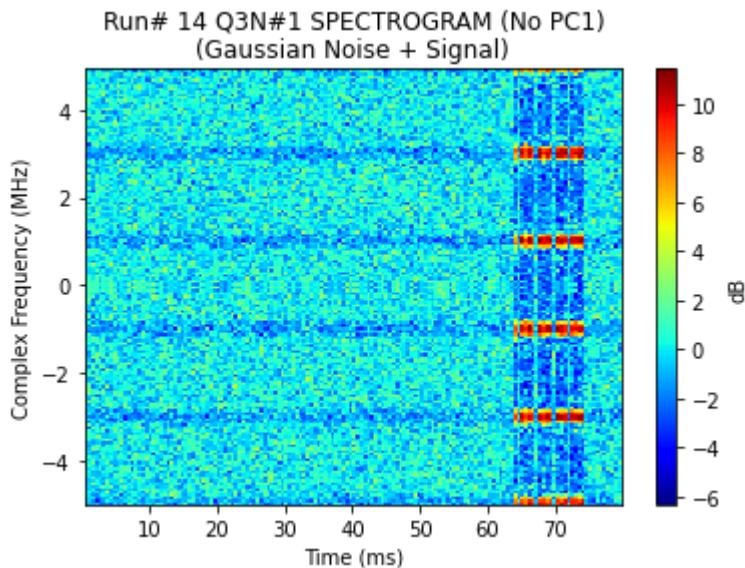




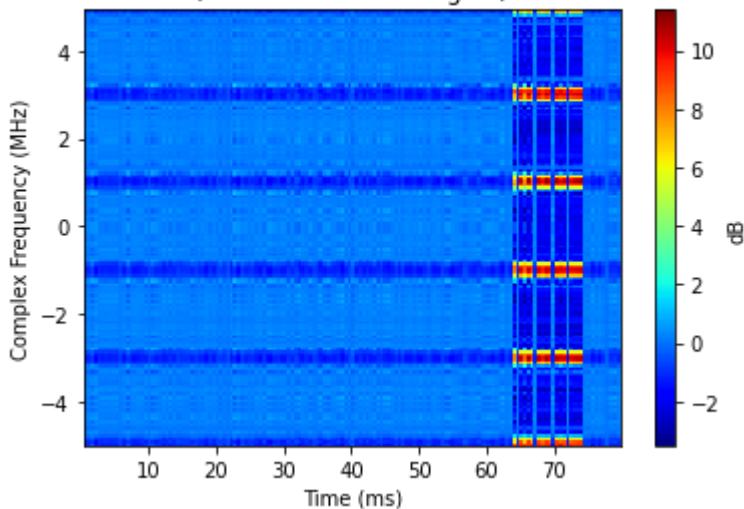




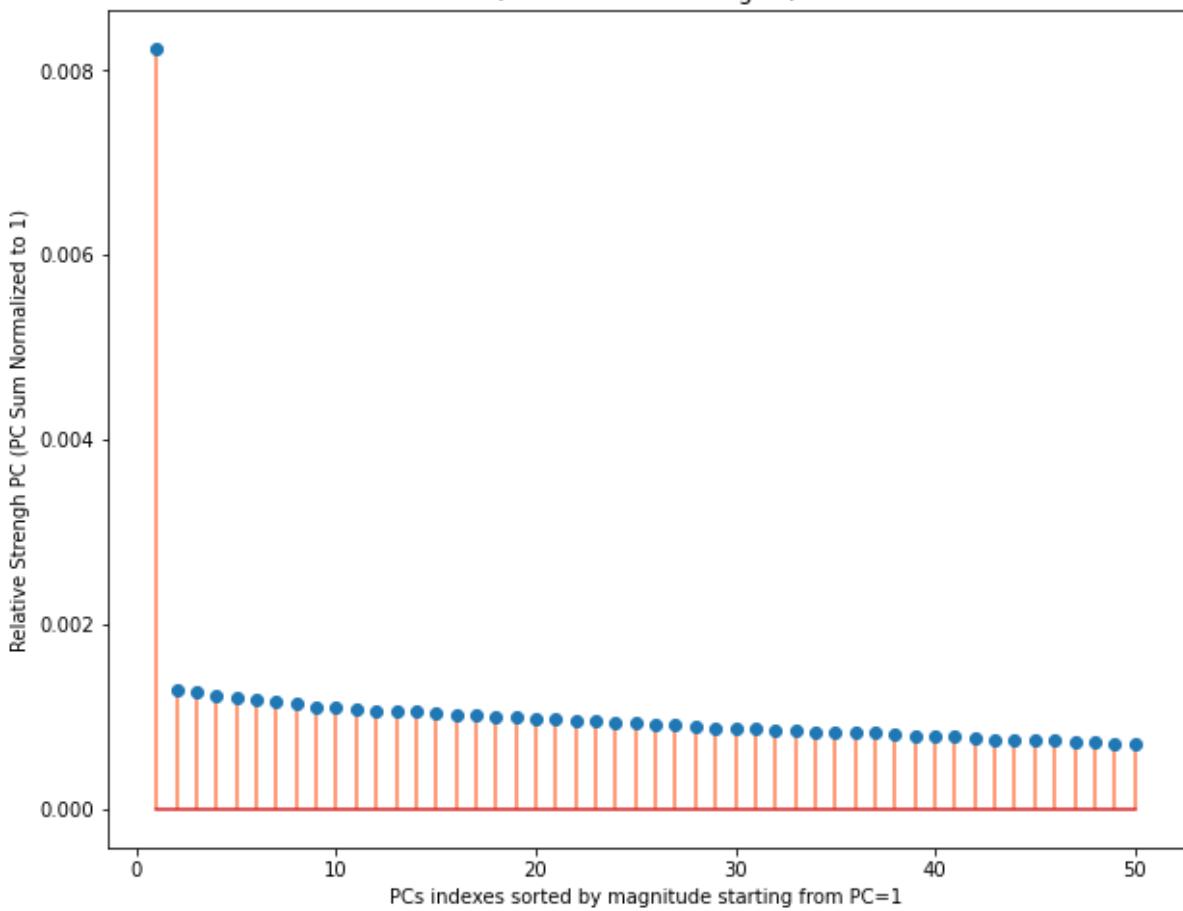


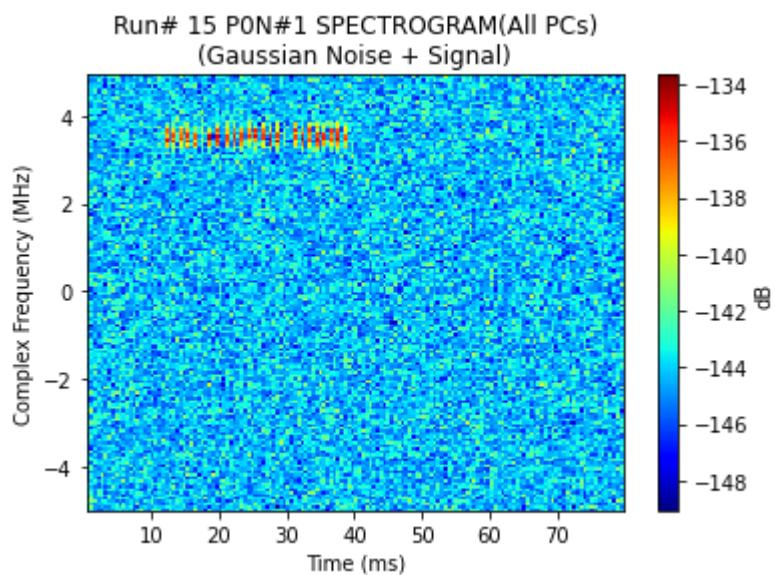
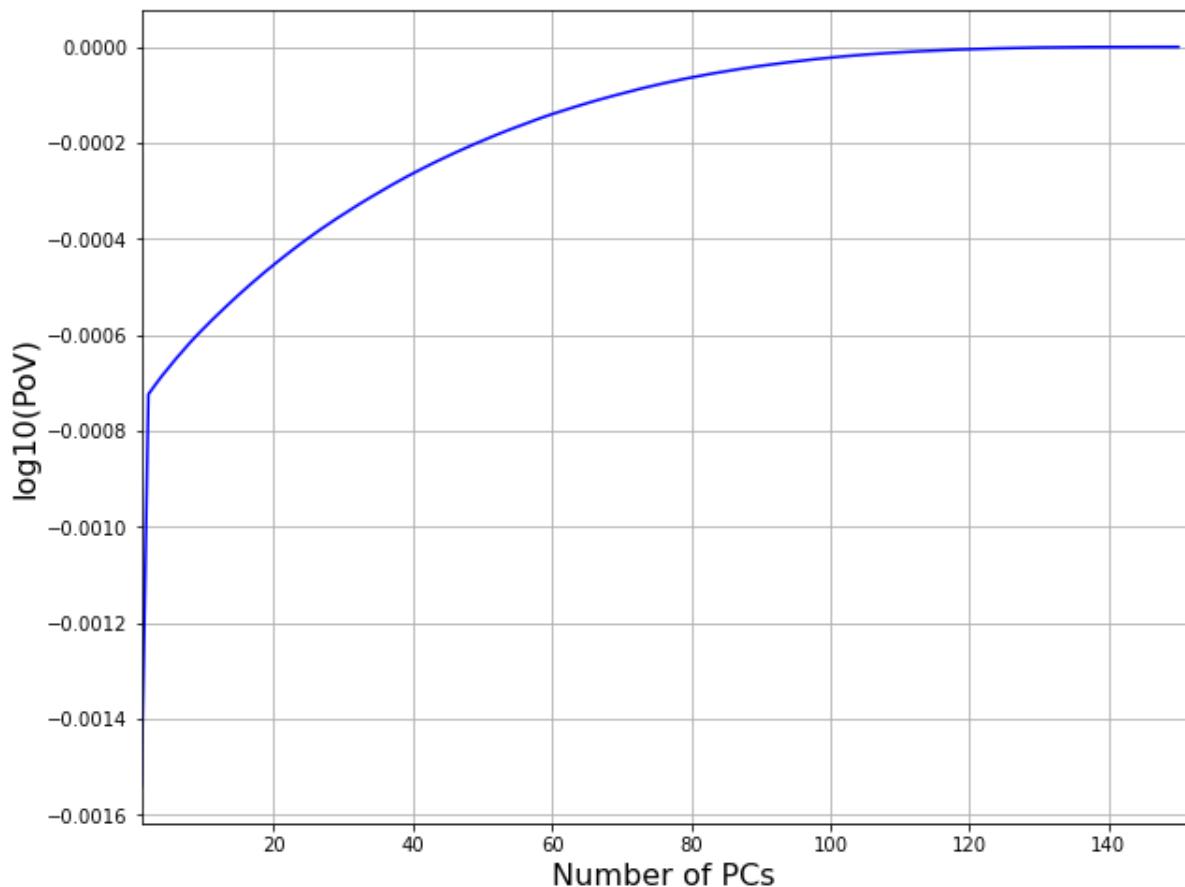


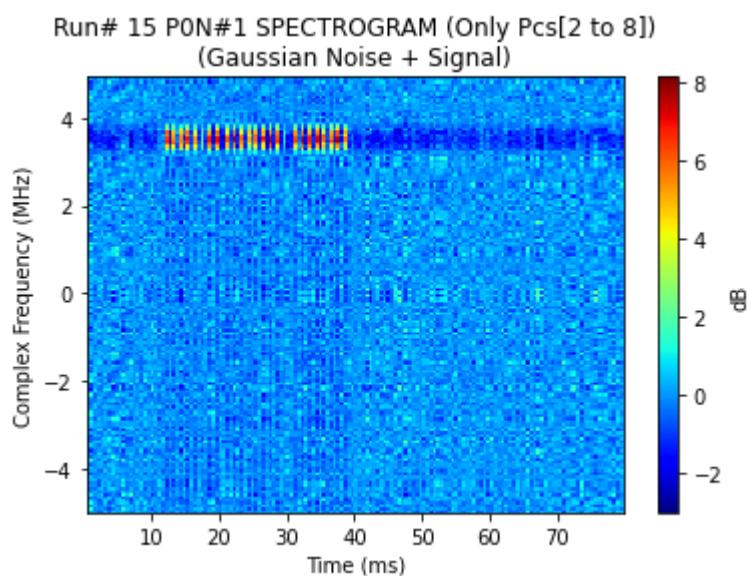
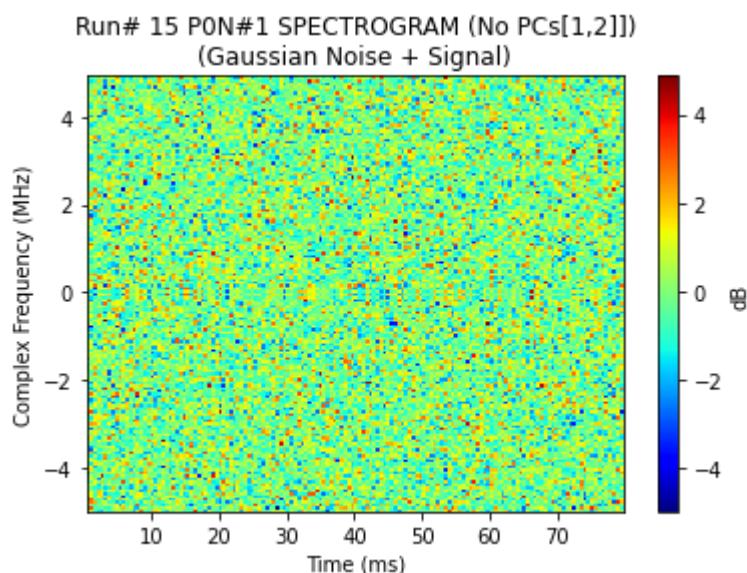
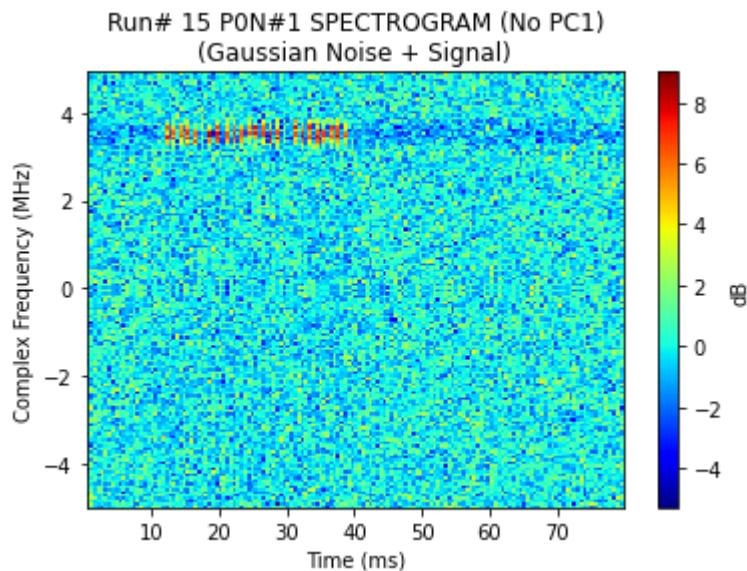
Run# 14 Q3N#1 SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise + Signal)



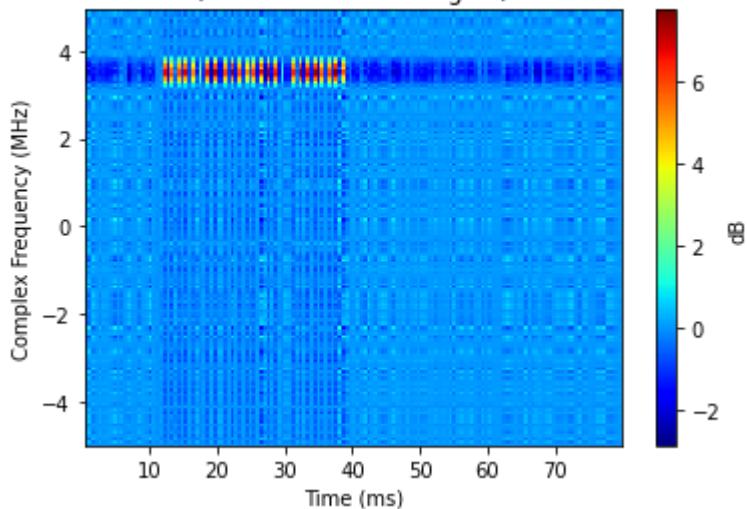
Run# 14 Q3N#1 PCs (50 largest, first/largest Not Shown).
NOTE: Plot Normalized: All PCs add to 1
(Gaussian Noise + Signal)



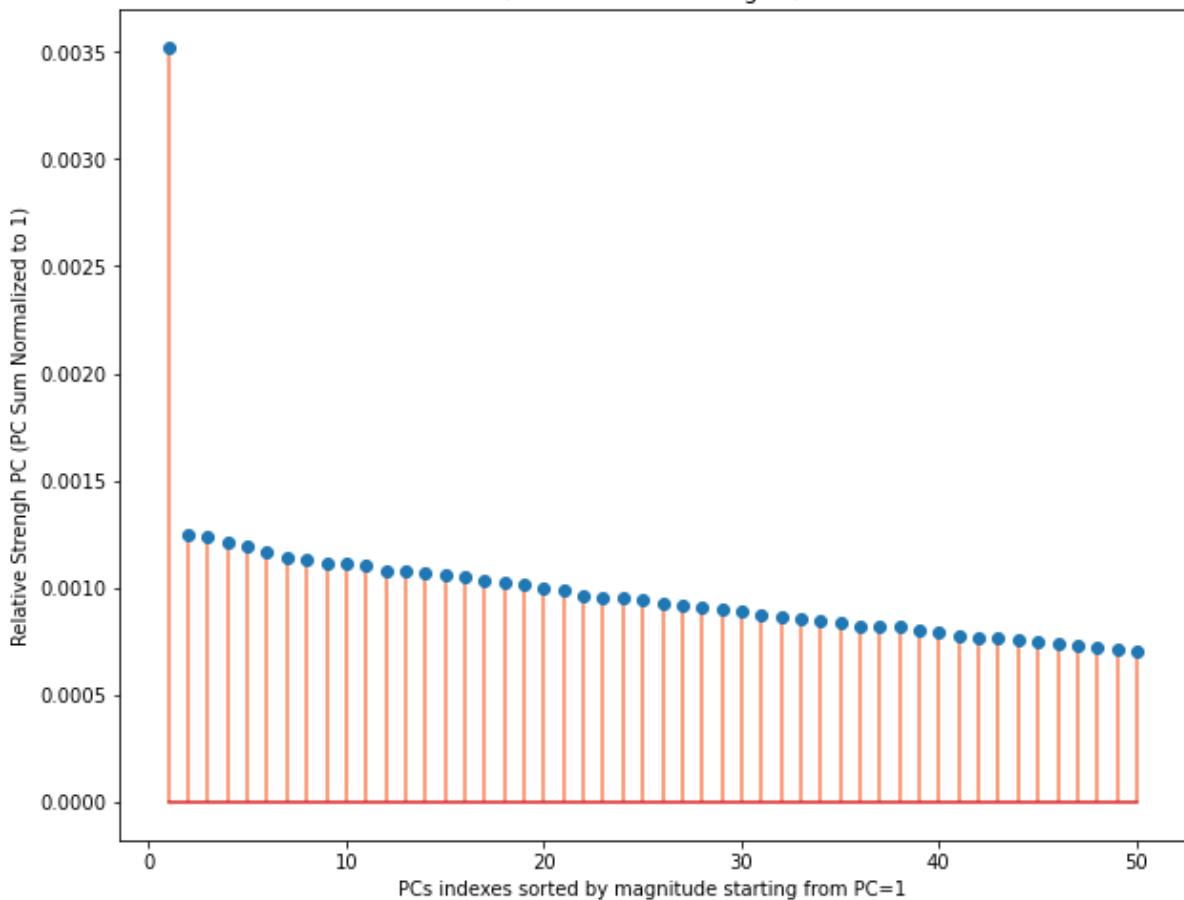


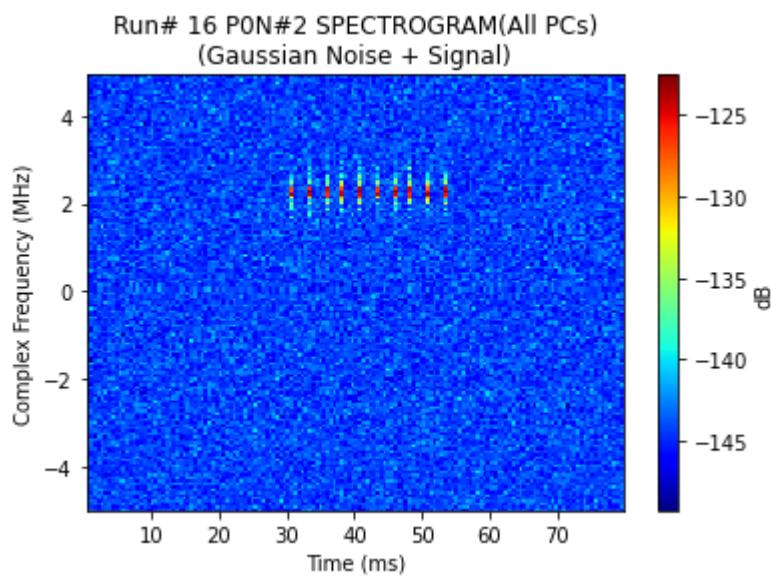
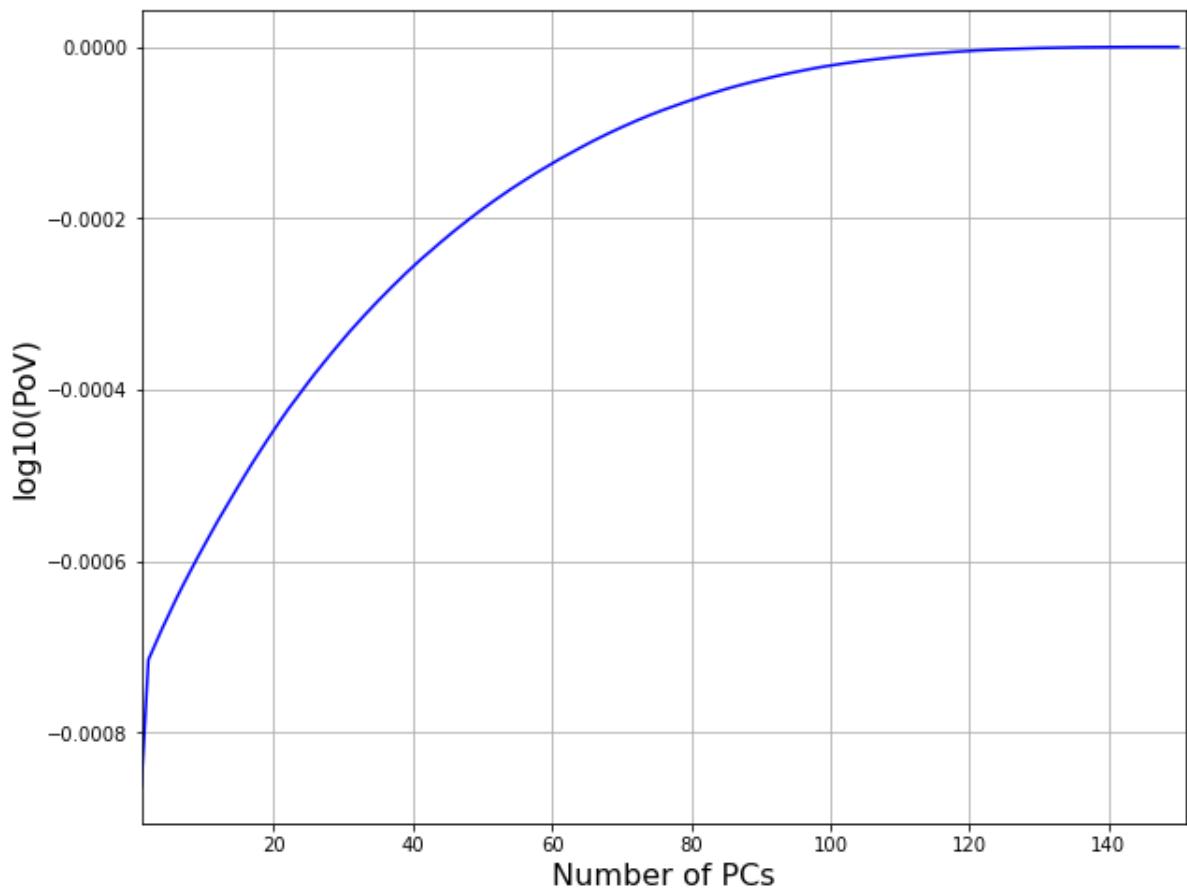


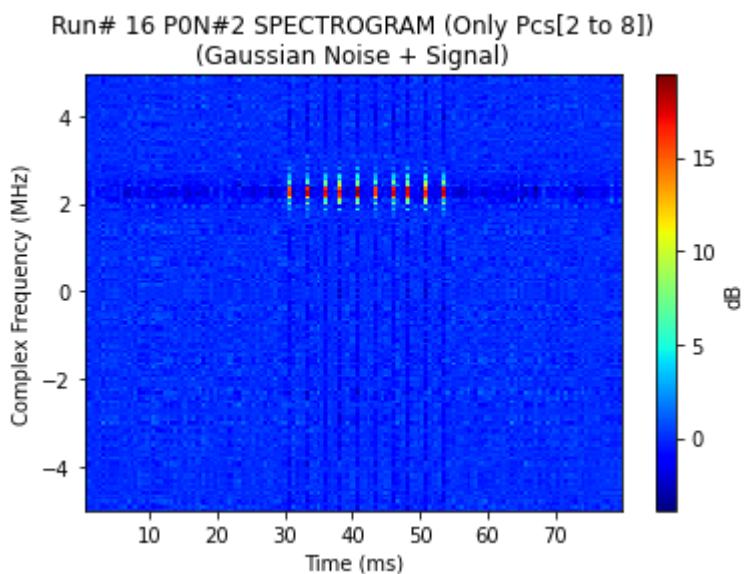
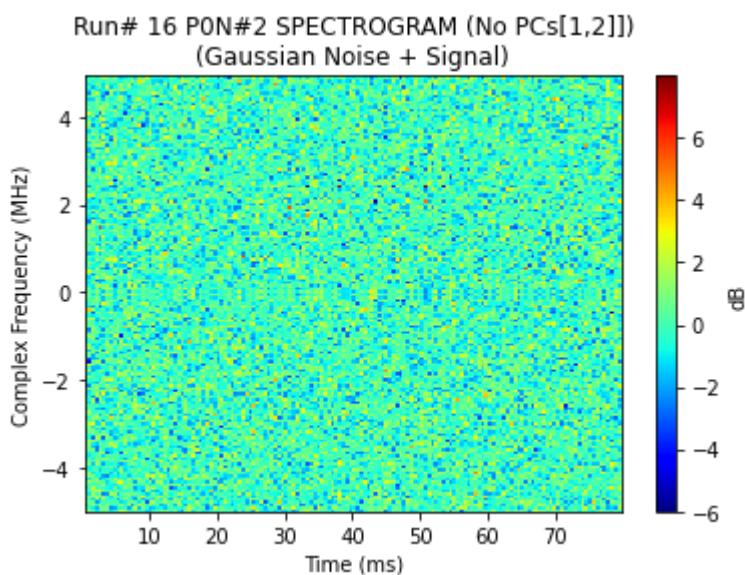
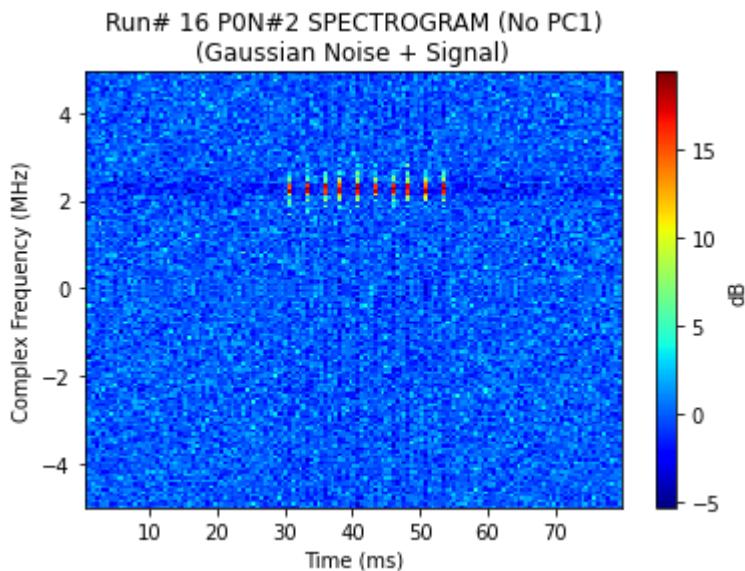
Run# 15 PON#1 SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise + Signal)



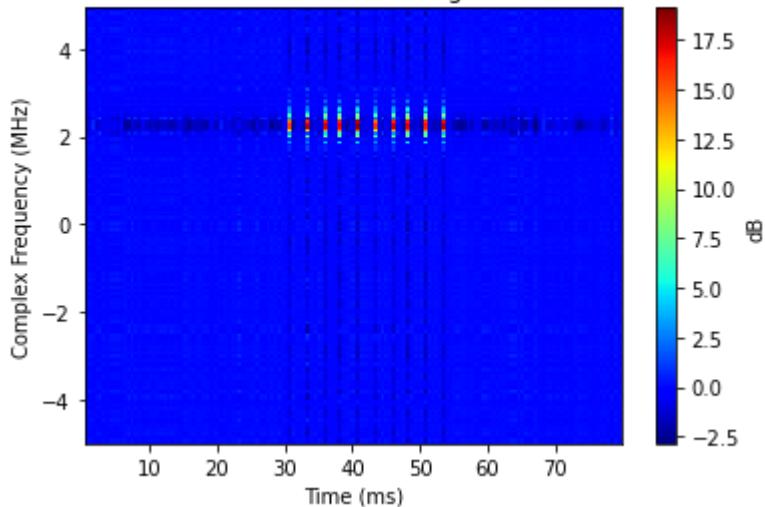
Run# 15 PON#1 PCs (50 largest, first/largest Not Shown).
NOTE: Plot Normalized: All PCs add to 1
(Gaussian Noise + Signal)



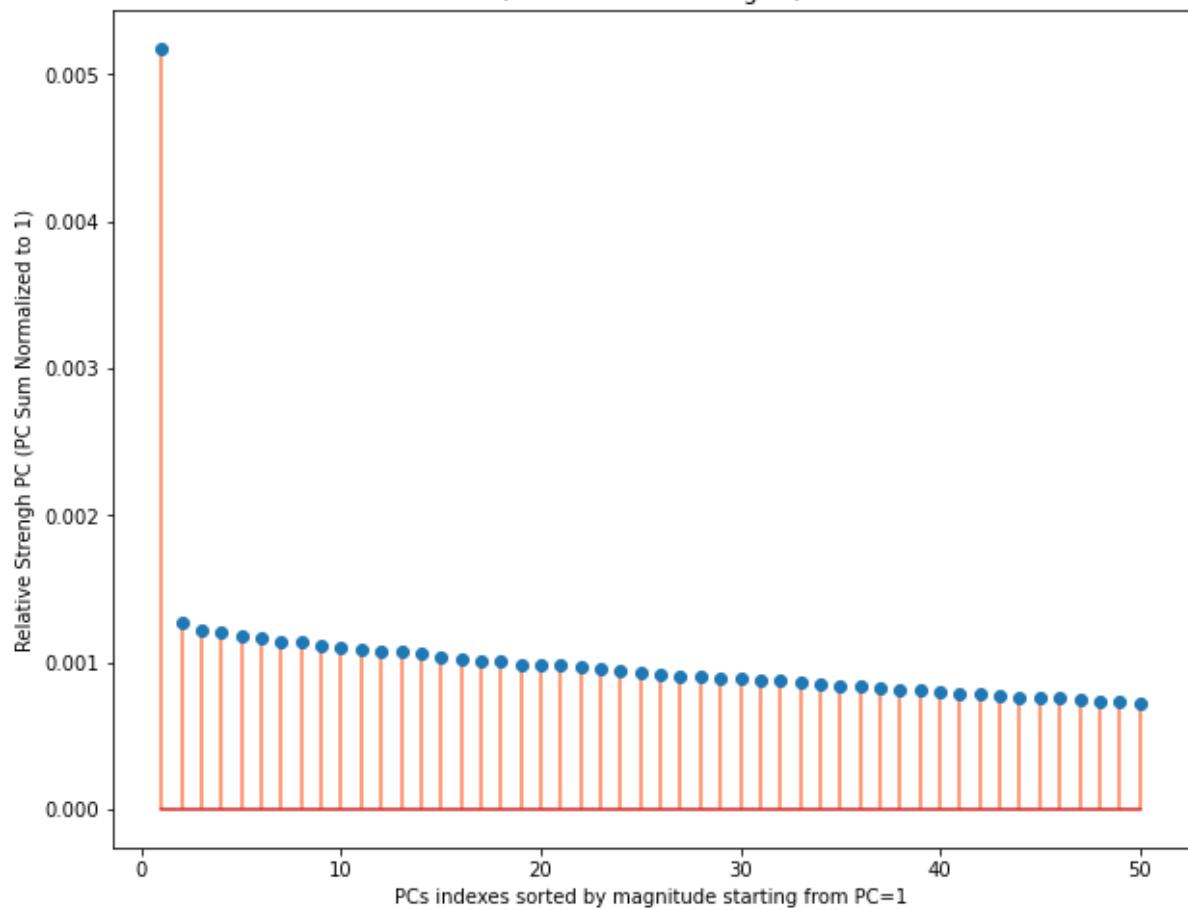


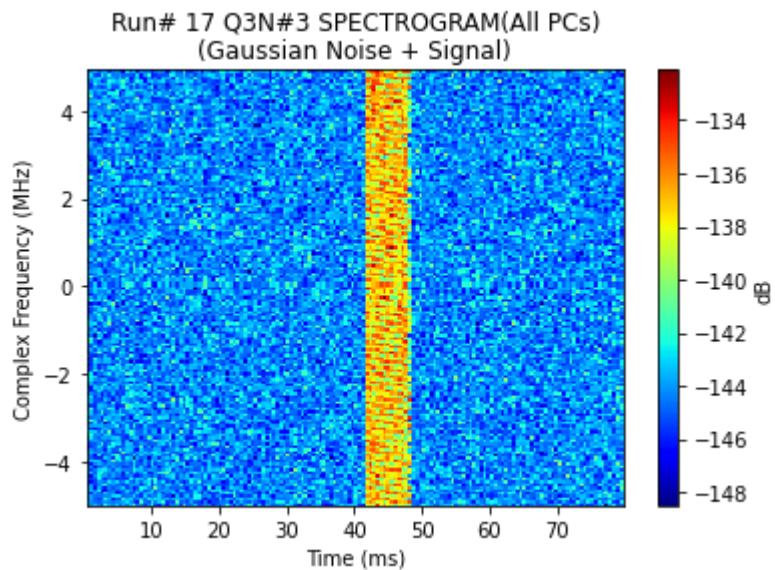
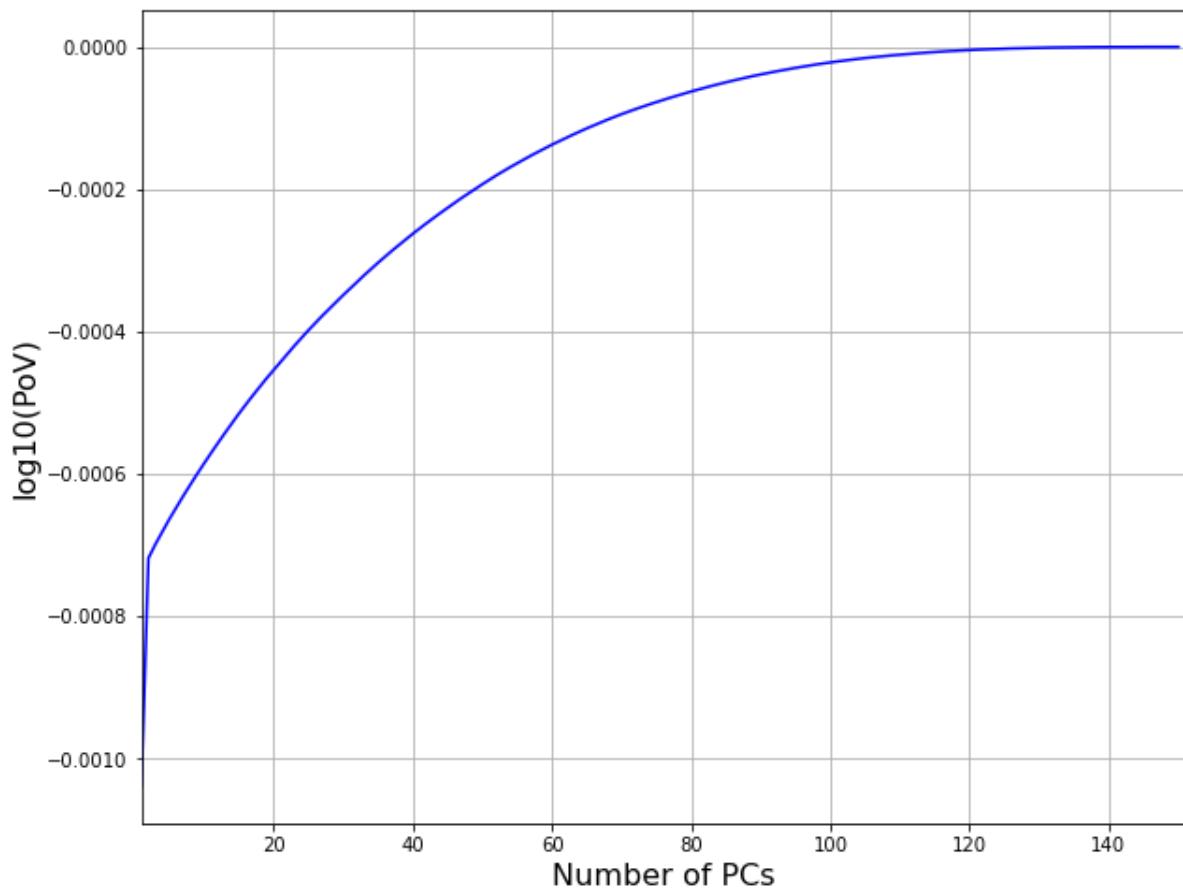


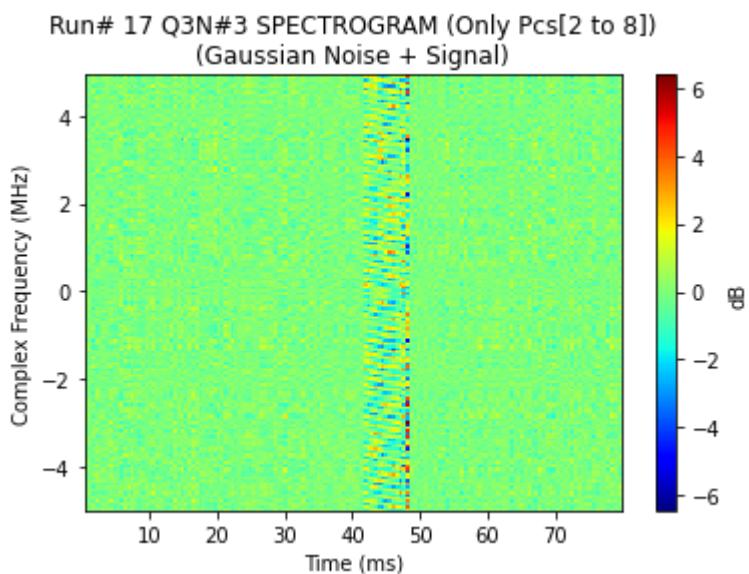
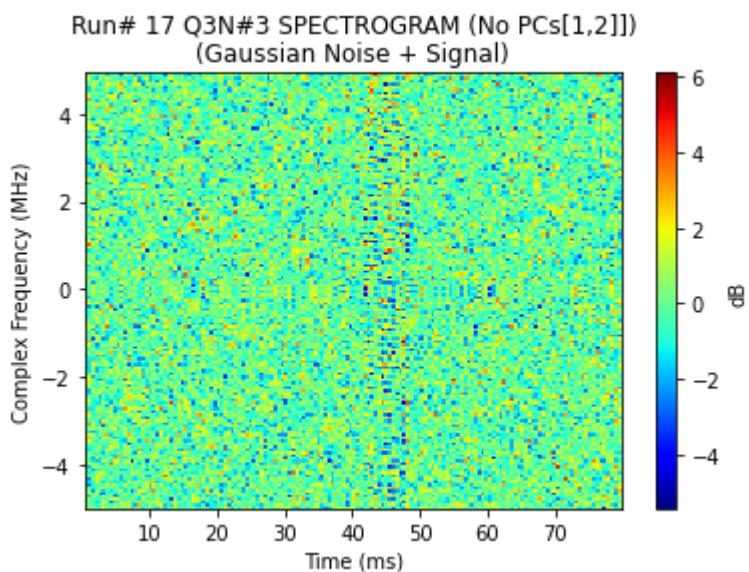
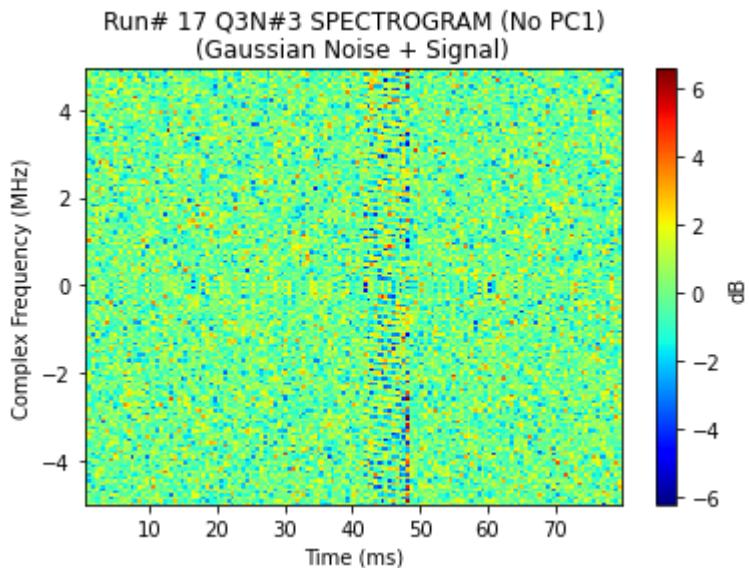
Run# 16 PON#2 SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise + Signal)



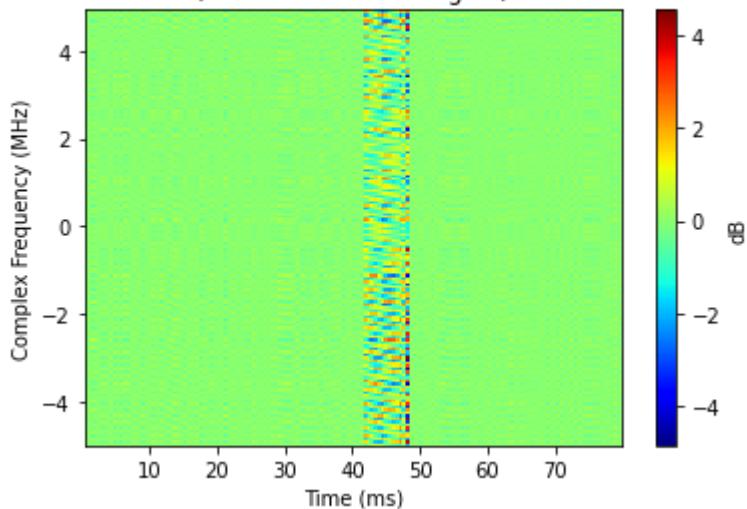
Run# 16 PON#2 PCs (50 largest, first/largest Not Shown).
NOTE: Plot Normalized: All PCs add to 1
(Gaussian Noise + Signal)



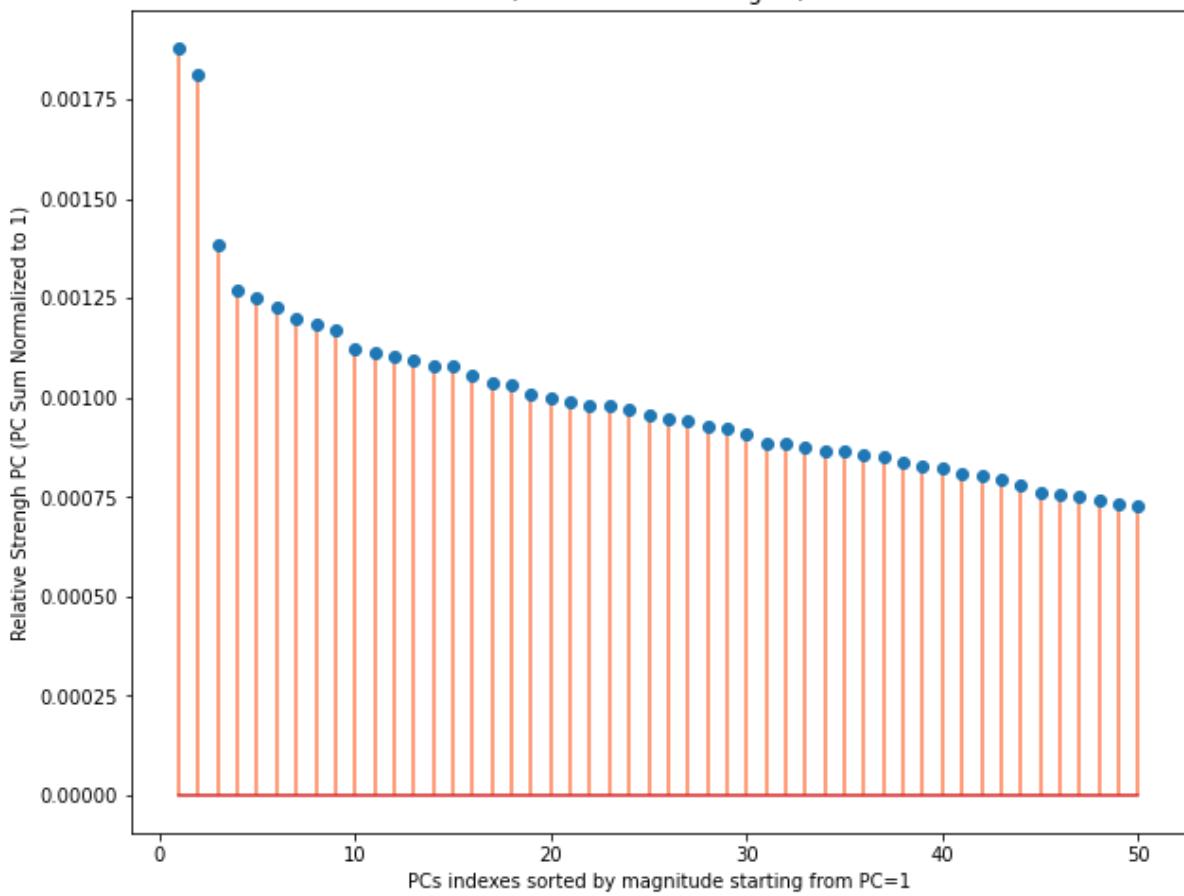


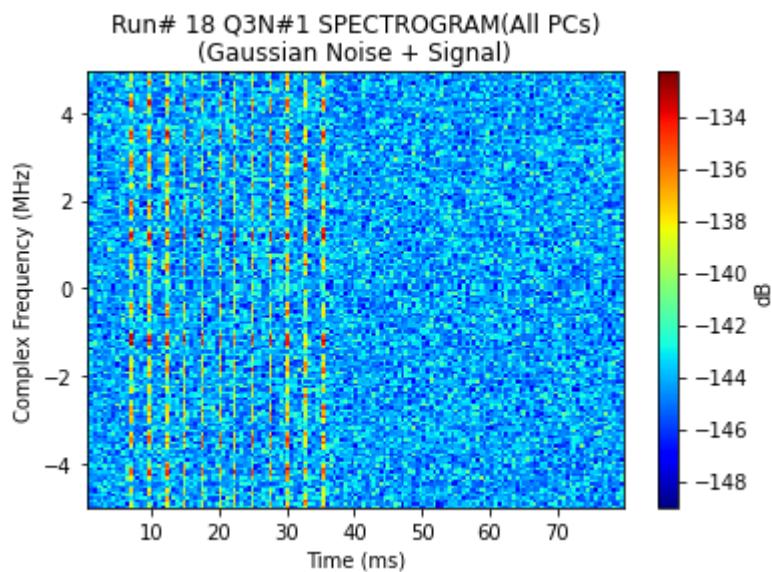
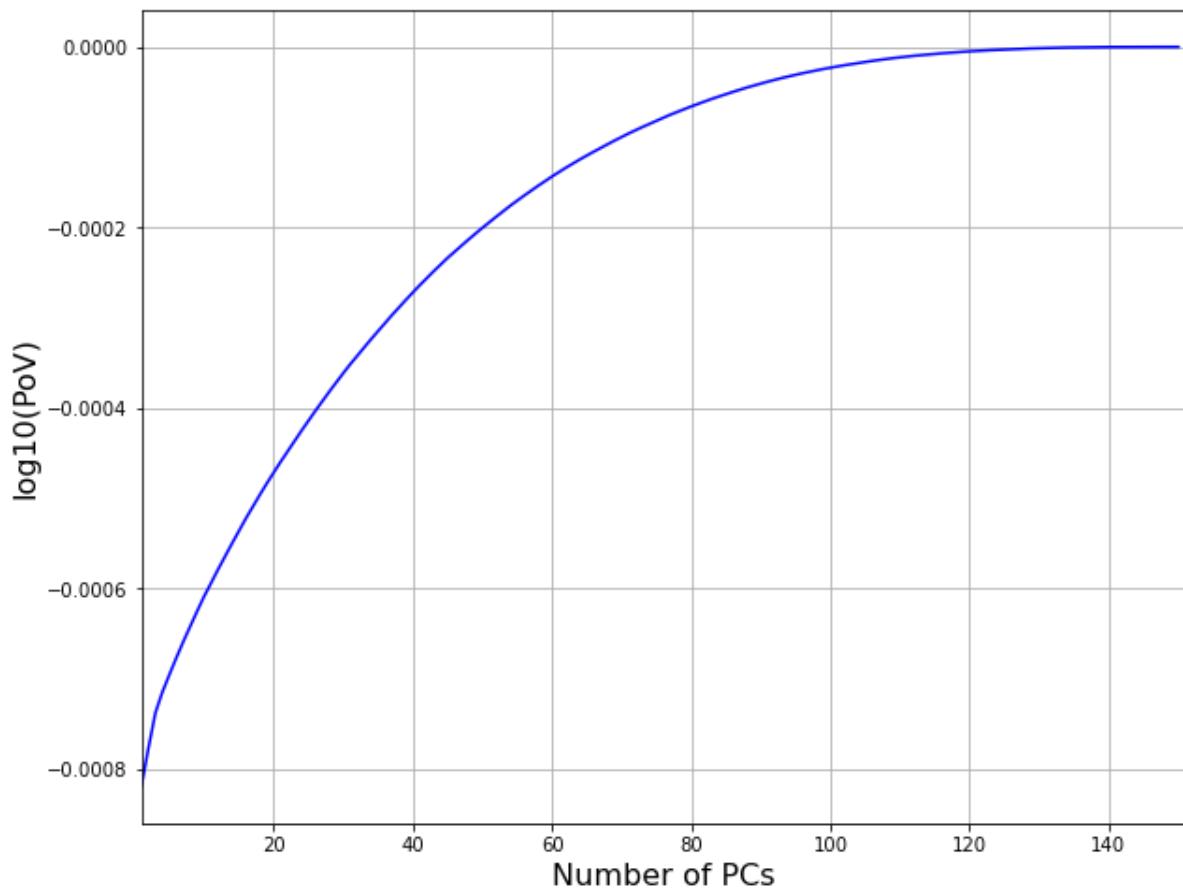


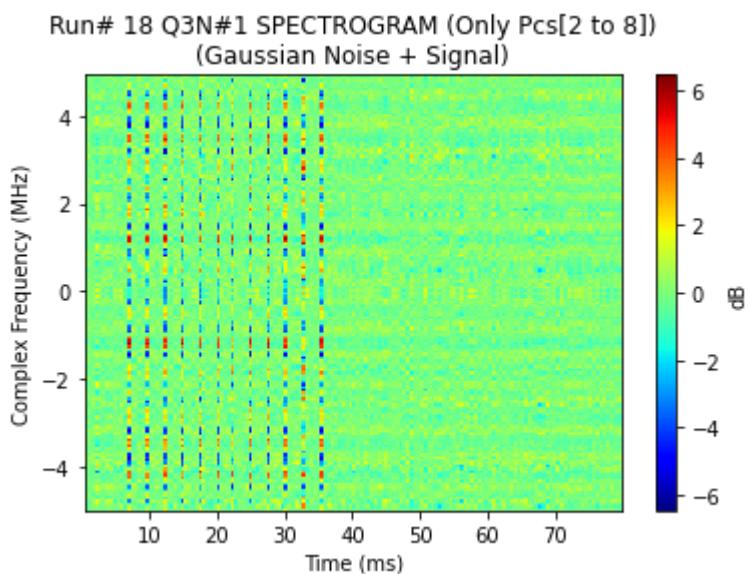
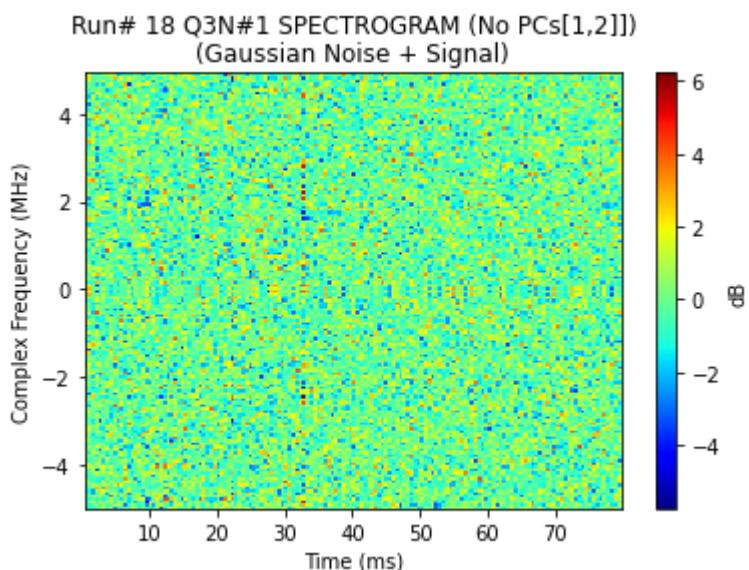
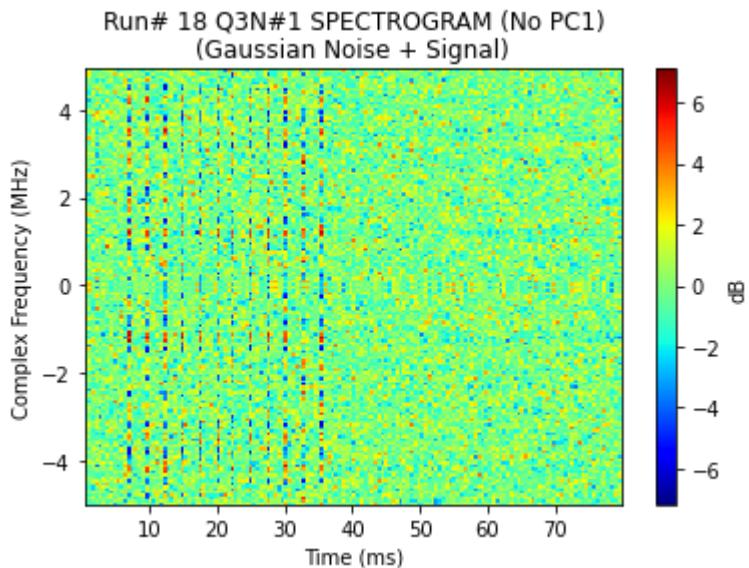
Run# 17 Q3N#3 SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise + Signal)



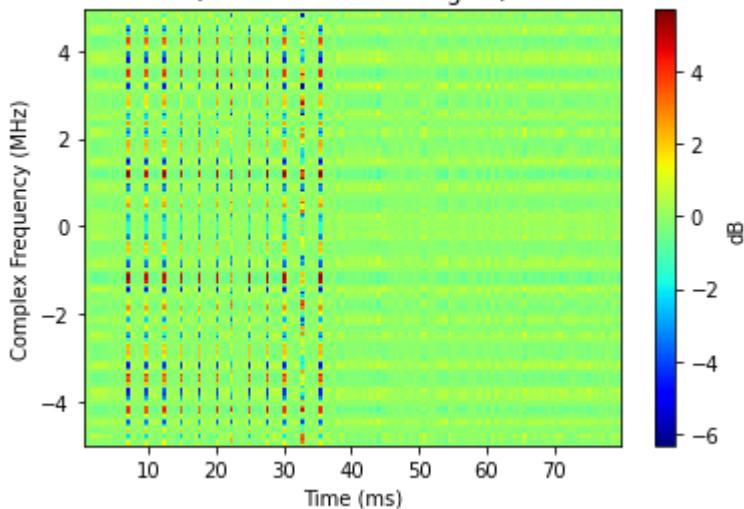
Run# 17 Q3N#3 PCs (50 largest, first/largest Not Shown).
NOTE: Plot Normalized: All PCs add to 1
(Gaussian Noise + Signal)



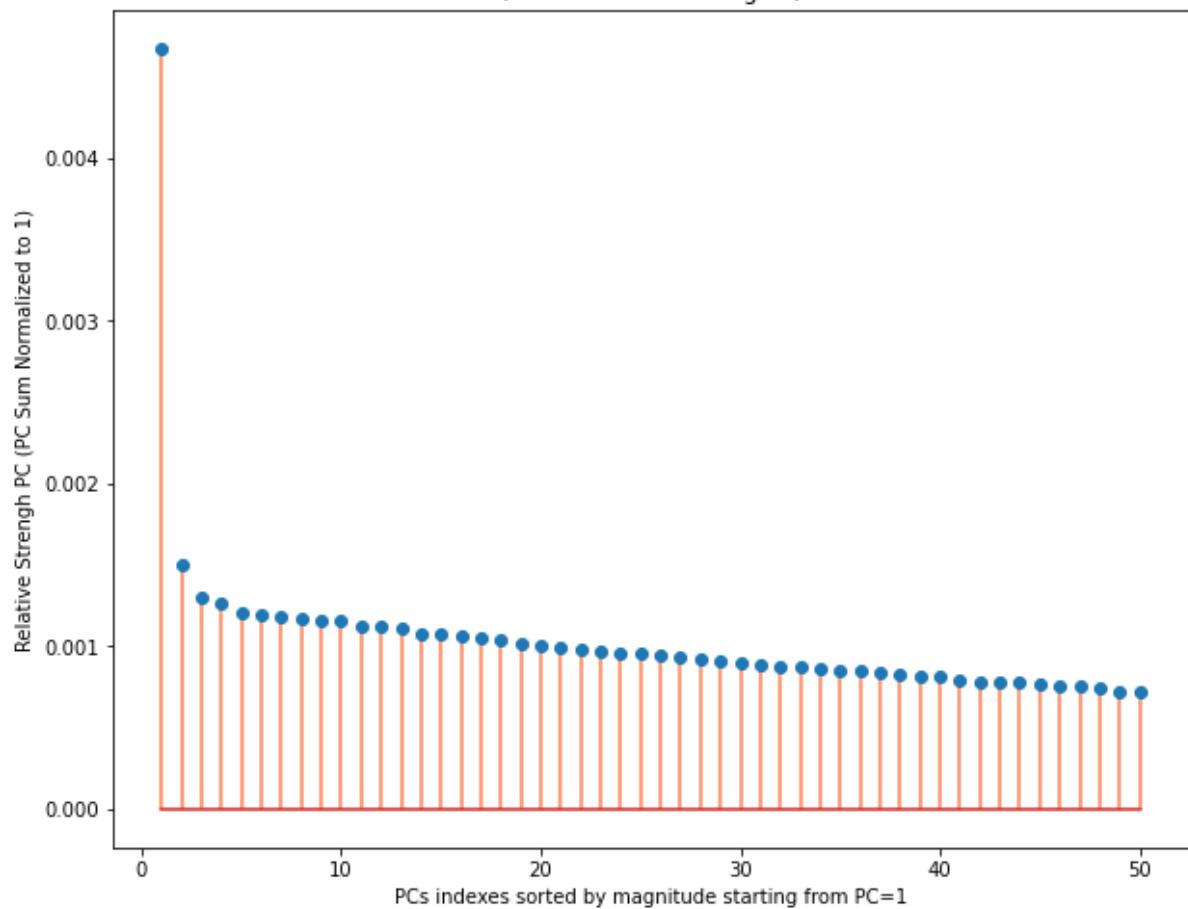


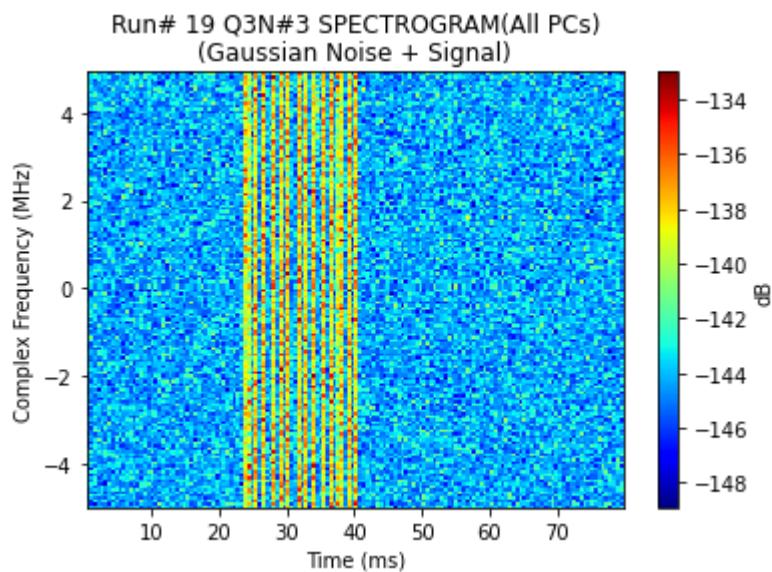
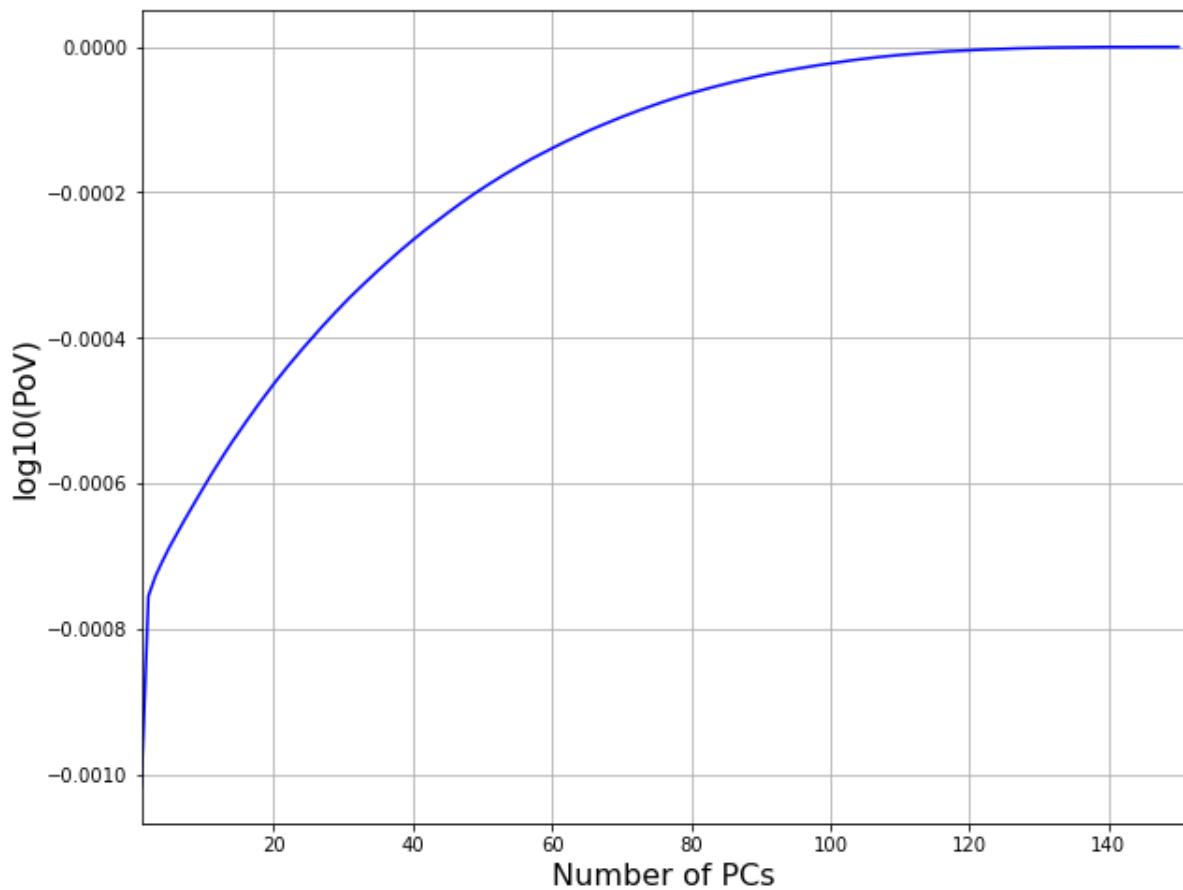


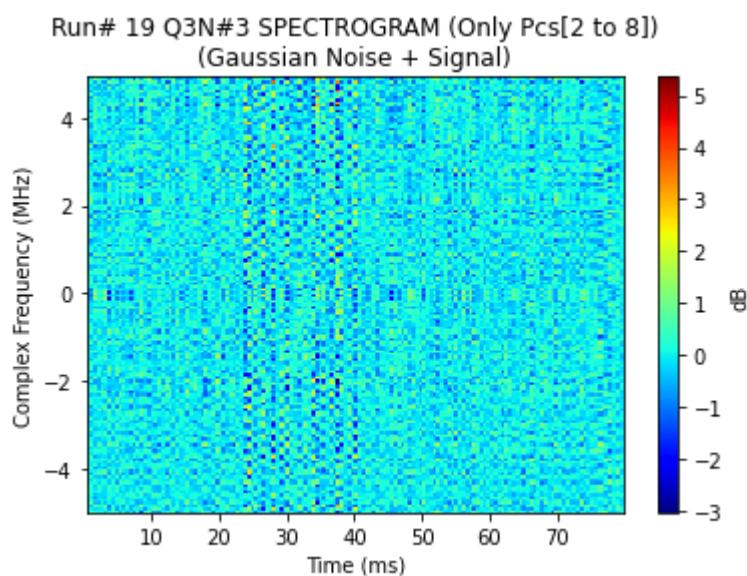
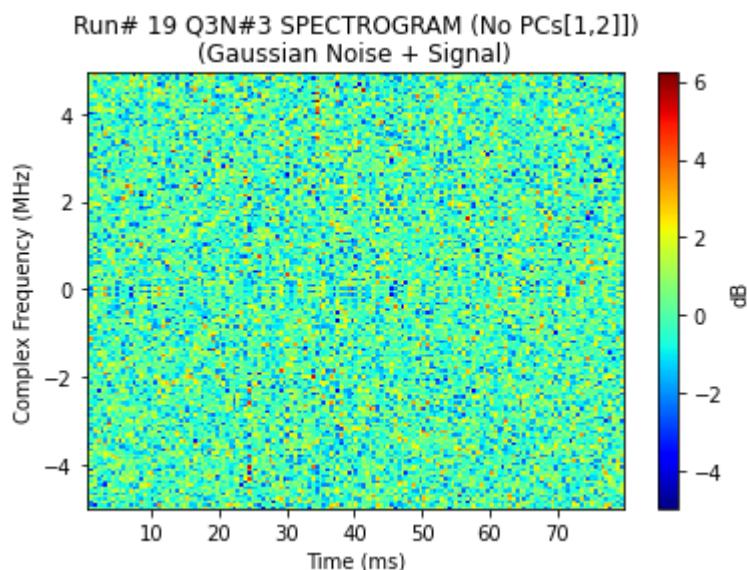
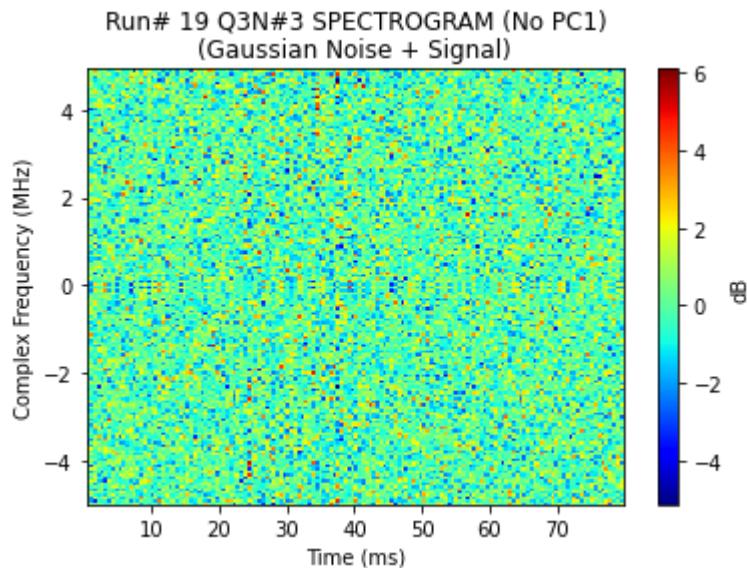
Run# 18 Q3N#1 SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise + Signal)

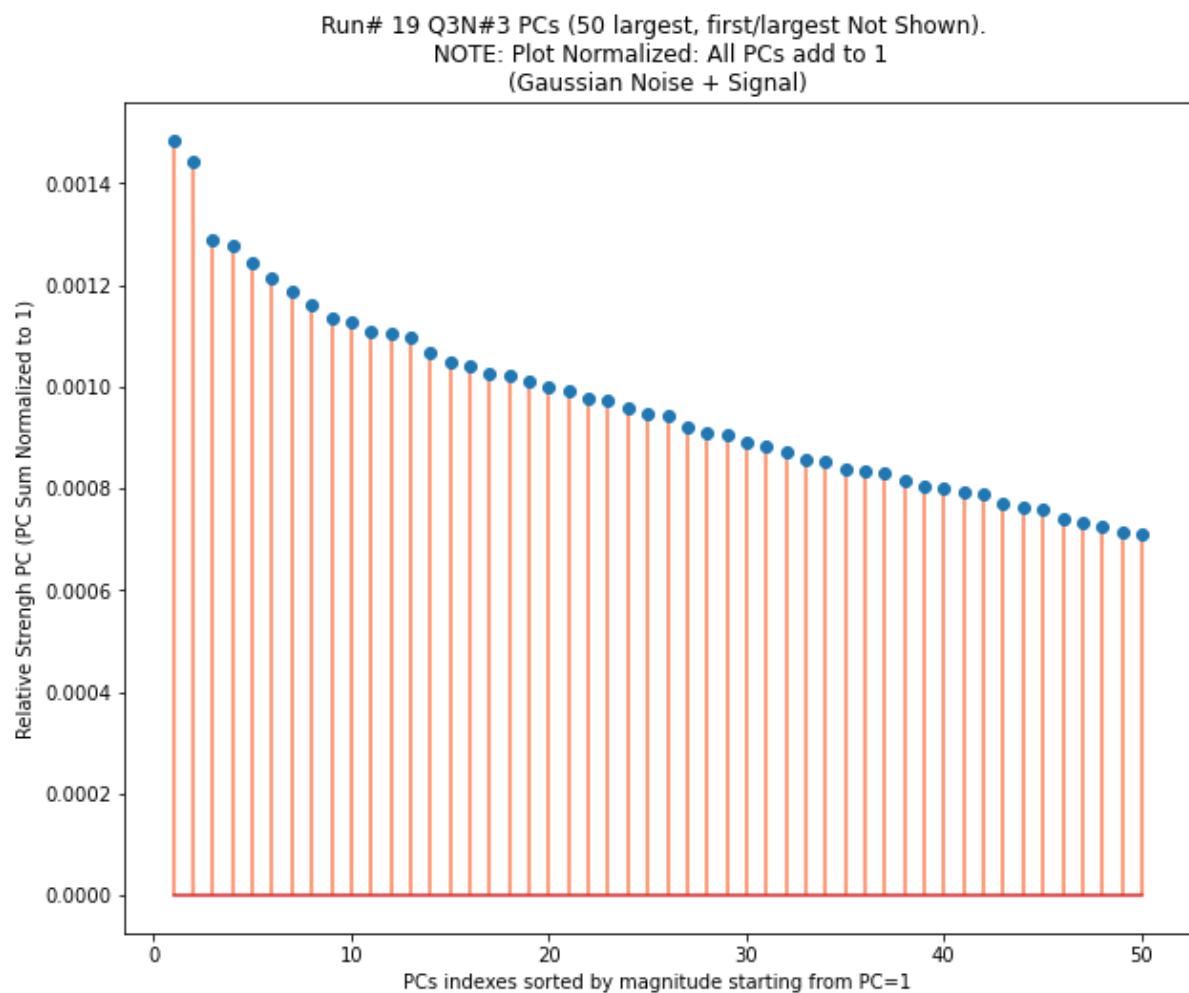
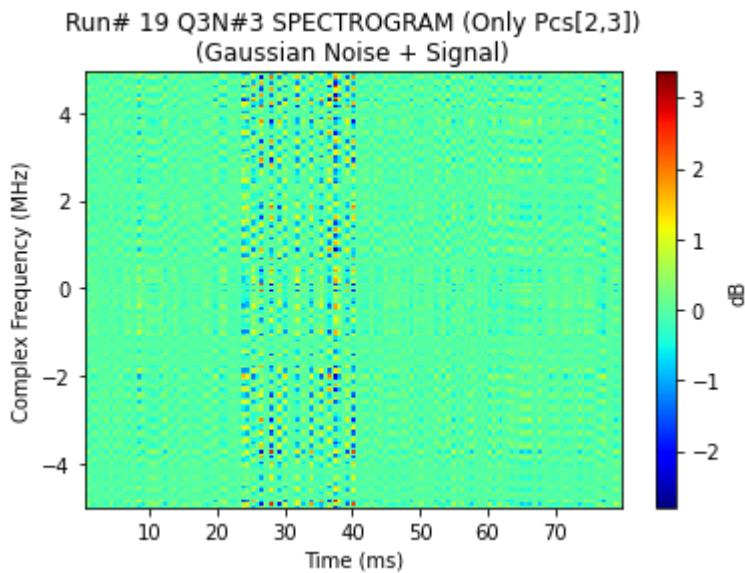


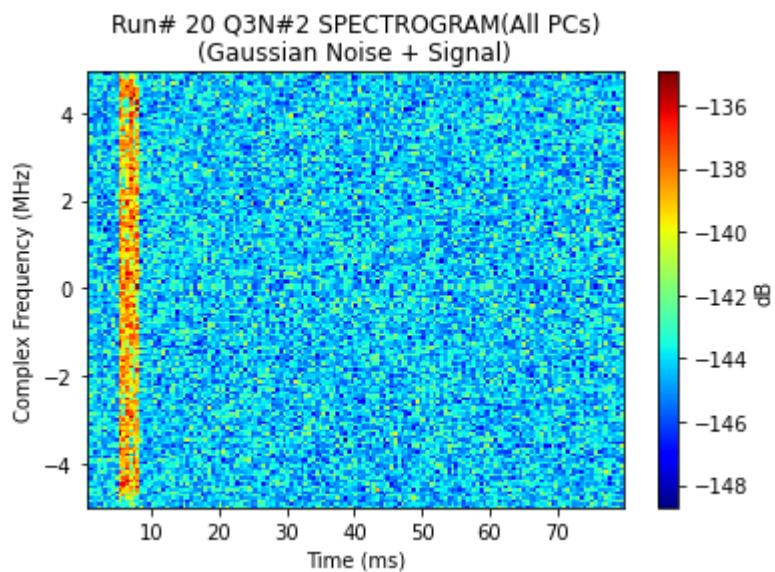
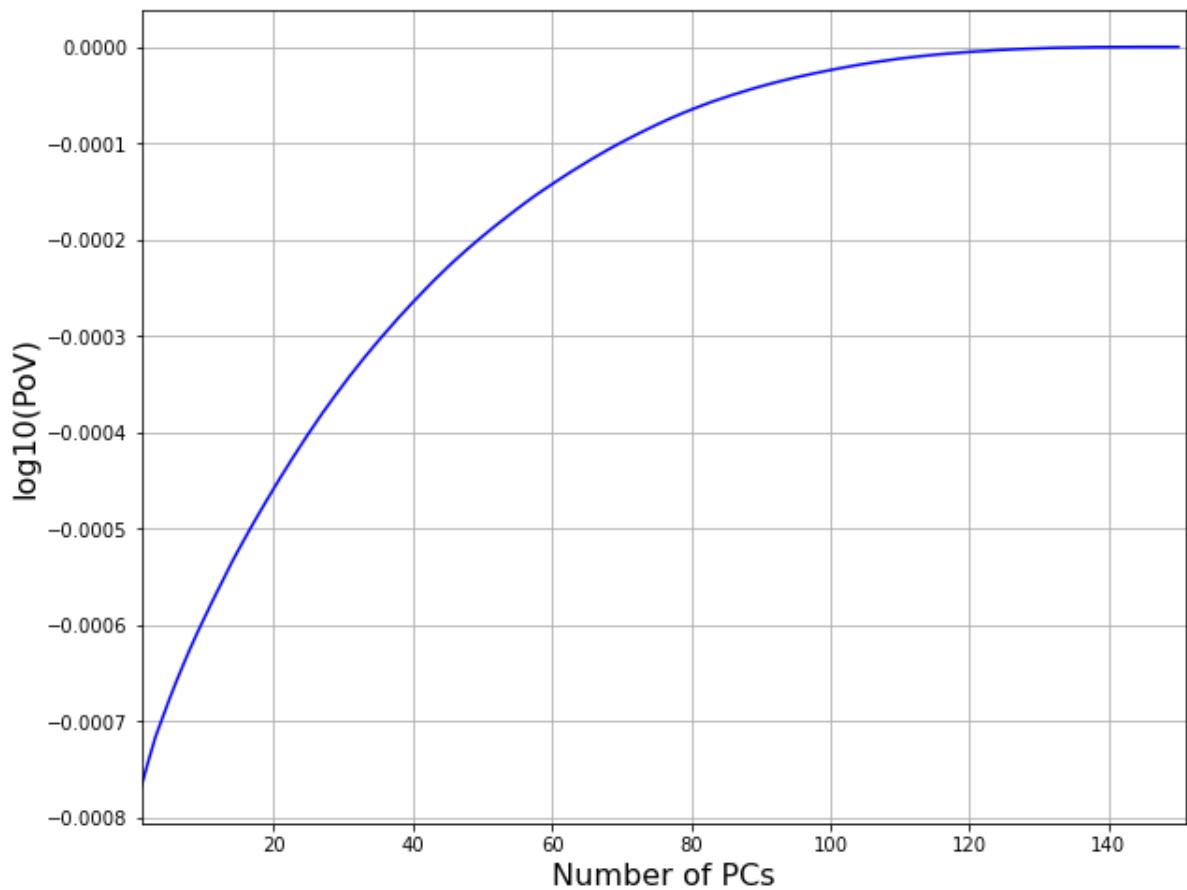
Run# 18 Q3N#1 PCs (50 largest, first/largest Not Shown).
NOTE: Plot Normalized: All PCs add to 1
(Gaussian Noise + Signal)

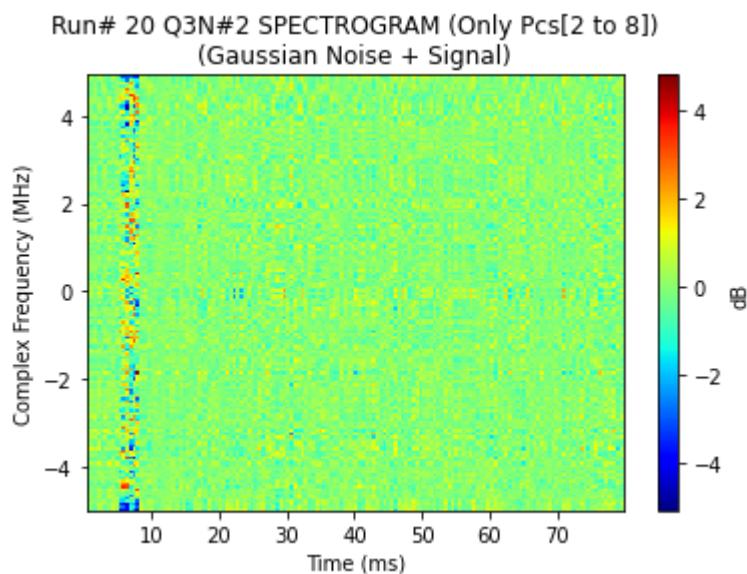
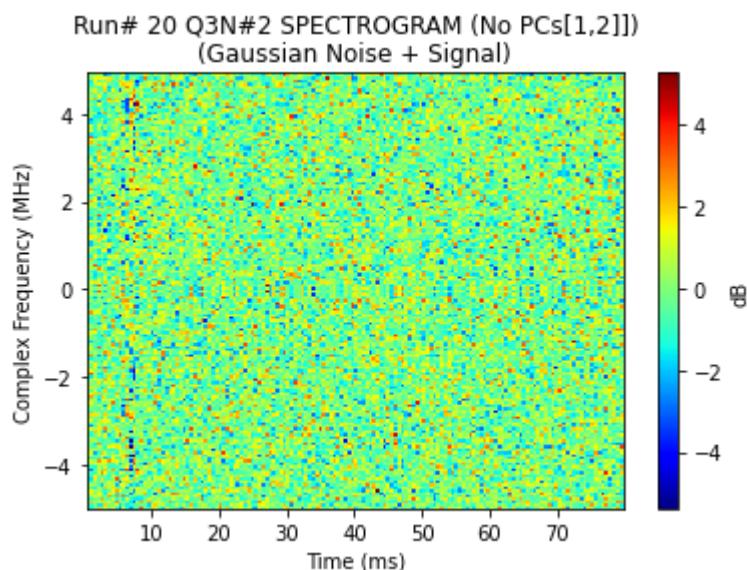
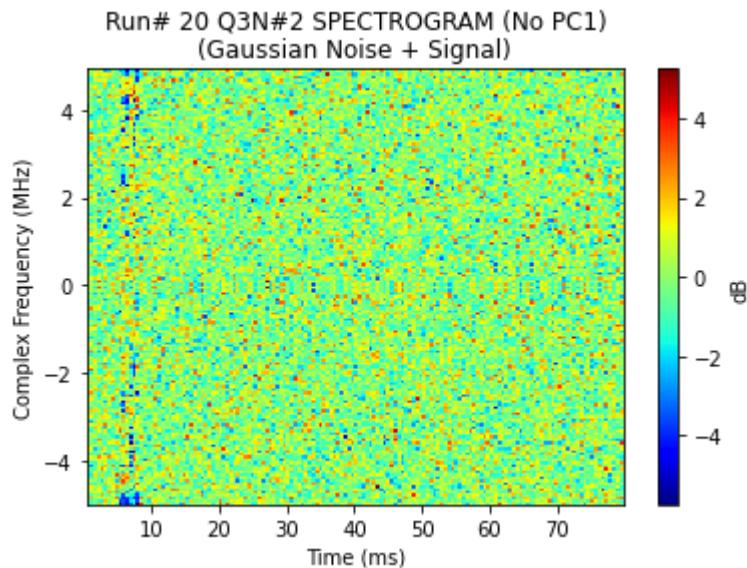




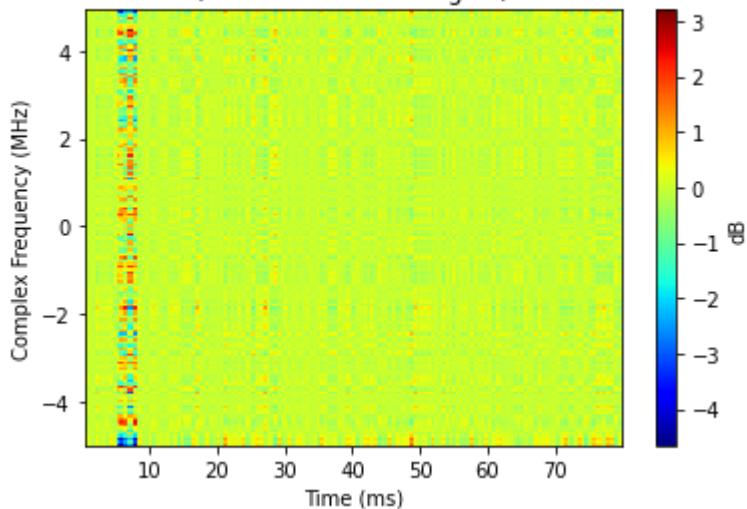




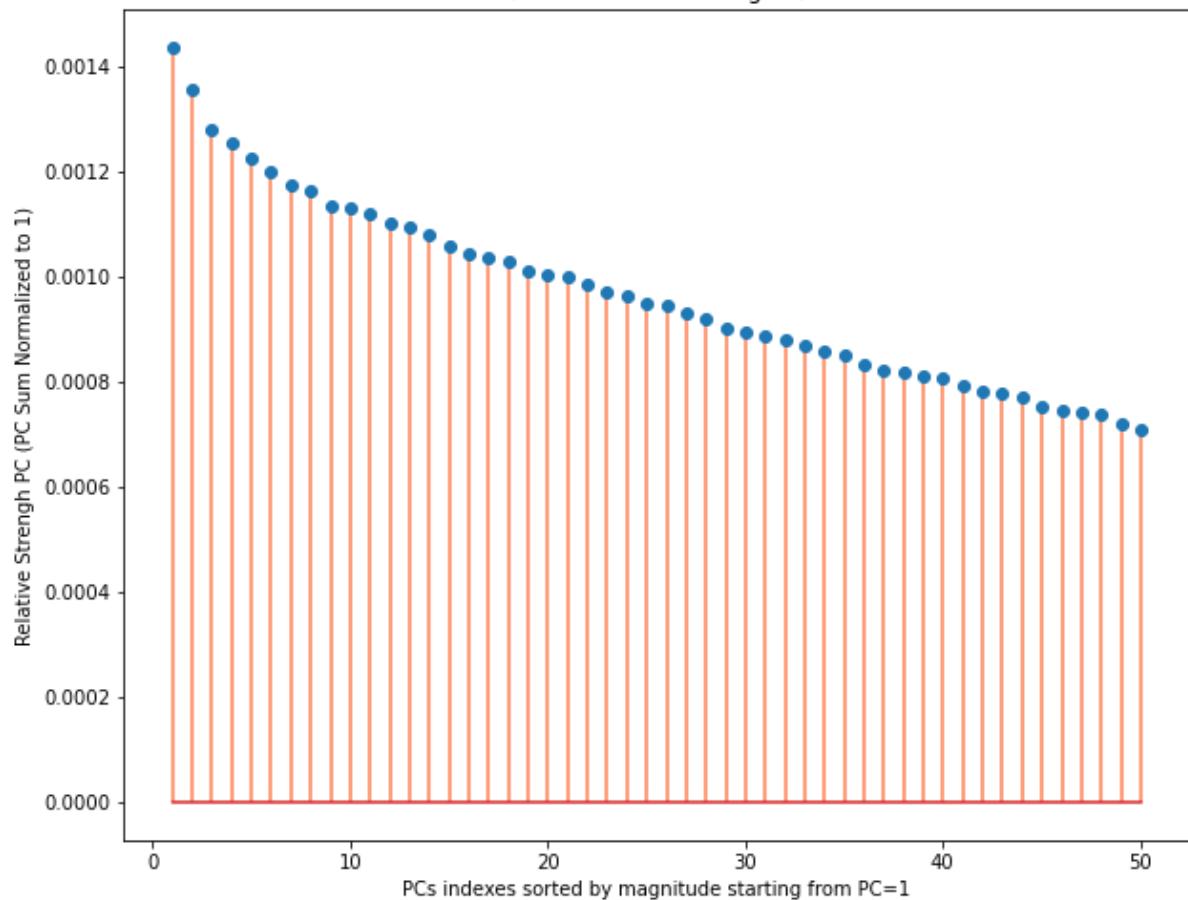


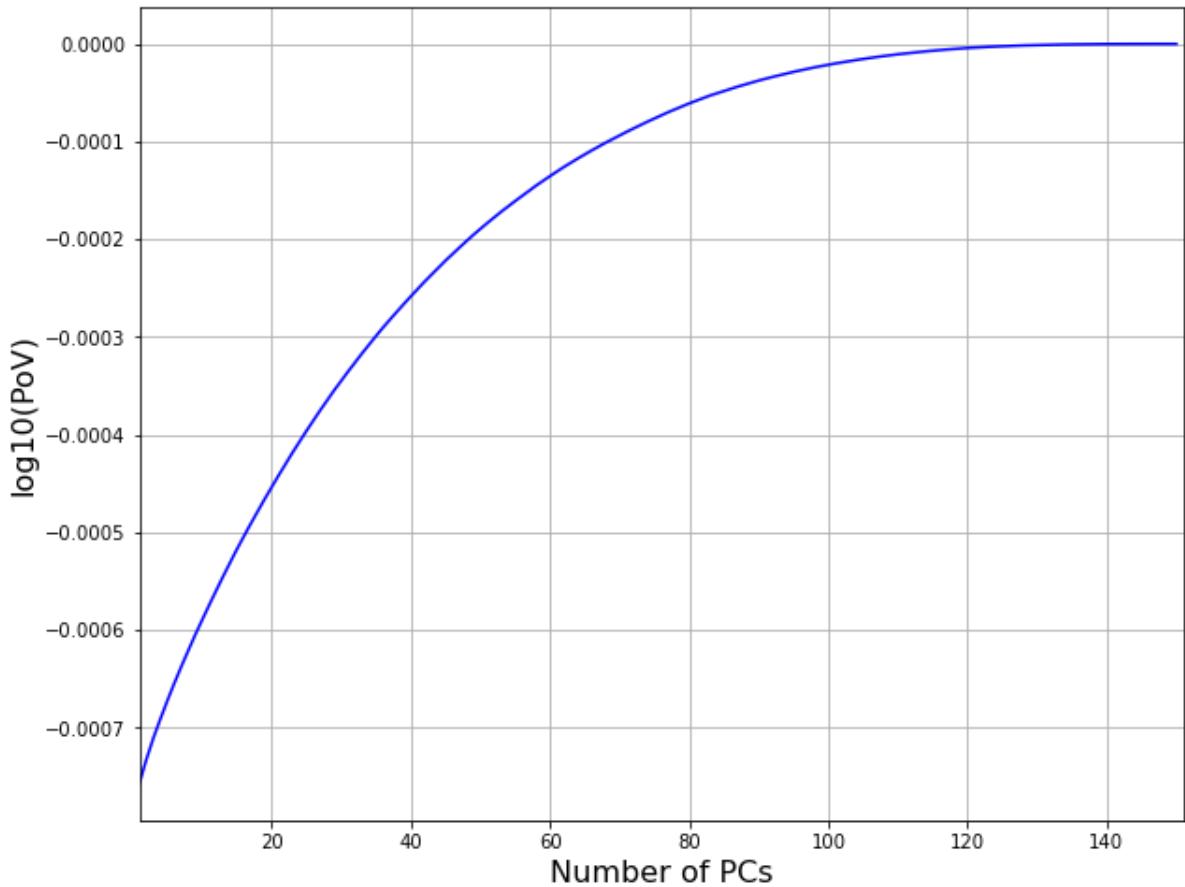


Run# 20 Q3N#2 SPECTROGRAM (Only Pcs[2,3])
(Gaussian Noise + Signal)



Run# 20 Q3N#2 PCs (50 largest, first/largest Not Shown).
NOTE: Plot Normalized: All PCs add to 1
(Gaussian Noise + Signal)





Now we will do a Logistic Regression to see if we can detect the presence of any of the waveform.

We will do the experiment for different numbers of PCs to see how the accuracy varies.

FIRST, we need calculate the PCs for all 200 samples. This will be our 'X' data. The y data will also be derived from the metafiles provided by NIST.

```
In [10]: nwav      = subsetSignals.shape[0]
MAT_Shi     = np.zeros([nwav, Nfft,Nfft])
MAT_U       = np.zeros([nwav, Nfft,Nfft])
MAT_S       = np.zeros([nwav, Nfft])
MAT_Vh      = np.zeros([nwav, Nfft,Nfft])

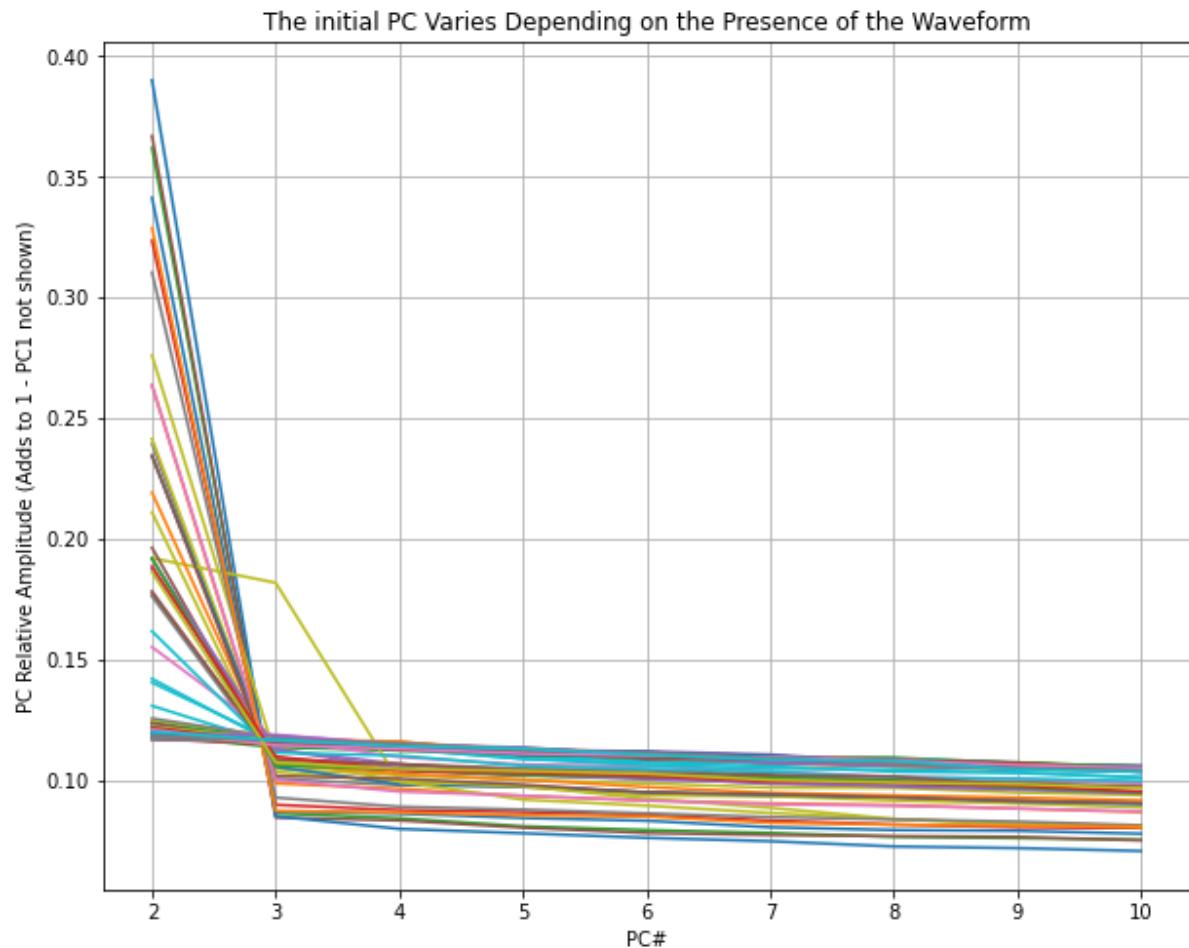
for idx in range(nwav):
    #print('****-----')
    #print('***Doing for idx=', idx)
    #print('...SNR   :', subsetInfo['SNR'][idx])
    #print('...BinNo :', subsetInfo['BinNo'][idx])
    f, t0, S0 = signal.spectrogram(subsetSignals[idx], fs=10e6, nperseg=Nfft, scaling='spectrum', return_onesided=False)
    S0[0:1, :] = S0[1:2, :]
    L = S0.shape[1]/groupby
    S1 = np.reshape(S0[:, :int(L)*groupby], (Nfft,int(L),groupby))
    S = npamax(S1, axis=-1)
    if flag_make_it_150x150: S=S[:,0:S.shape[0]] #Make is 150x150 Square
    t = t0[groupby-1::groupby]
    fshi = np.fft.fftshift(f/1e6)
    Sshi = np.fft.fftshift(10.*np.log10(S), axes=0)
    MAT_Shi[idx]= Sshi
    MAT_U[idx],MAT_S[idx],MAT_Vh[idx]=np.linalg.svd(Sshi,full_matrices=False);
    #print('...U.shape=', u.shape)
    #print('...S.shape=', s.shape)
    #print('...Vh.shape', vh.shape)
```

Here we print the PCs. Note that there is variability to exploit, specially in the second component.

We will try to exploit this variability to make predictions.

In [11]: MINPCA=1
MAXPCA=10

```
plt.figure(figsize=(10,8))
for idx in range(nwav//2):
    #plt.stem(MAT_S[idx,MINPCA:MAXPCA]/np.sum(MAT_S[idx,MINPCA:MAXPCA]), use_line_collection=True)
    plt.plot(np.arange(MINPCA+1,MAXPCA+1),MAT_S[idx,MINPCA:MAXPCA]/np.sum(MAT_S[idx,MINPCA:MAXPCA]))
plt.grid()
plt.title("The initial PC Varies Depending on the Presence of the Waveform")
plt.xlabel("PC#");
plt.ylabel("PC Relative Amplitude (Adds to 1 - PC1 not shown)");
```



Now we will do the Logistic Regression to detect the signal (binary prediction). Because we are using only 200 samples (a file with 200 samples is 2.5 GB large), we will use KFolding.

We will assess many different numbers of PCs to see if more PCs help.

```
In [12]: #from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold

y = subsetRadarStatus.ravel()
X = MAT_S

# Create a K-fold object
nfold = 5
kf = KFold(n_splits=nfold, shuffle=True)
kf.get_n_splits(X)

# Number of PCs to try
ncomp_test = np.array([1,2,3,4,5,6,7,8,9,10,20,30,40,50,100,X.shape[1]])
#ncomp_test = np.arange(2,2 +1)
num_nc = len(ncomp_test)
print("We will try %d # of PCs: "%num_nc, ncomp_test)
print('X.shape=', X.shape)

# Accuracy: acc[icomp,ifold] is test accuracy when
# using `ncomp = ncomp_test[icomp]` in fold `ifold`.
acc = np.zeros((num_nc,nfold))
acc_comp = np.zeros((num_nc))

# Loop over number of components to test
for icomp, ncomp in enumerate(ncomp_test):
    print(":::-----")
    print(":::Trying (icomp,ncomp) = (", icomp, ',', ncomp, ')')
    # Look over the folds
    for ifold, I in enumerate(kf.split(X)):
        Itr, Its = I

        # Xtr, Xts, ytr, yts = ...
        Xtr = X[Itr, 0:ncomp]; ytr = y[Itr];
        Xts = X[Its, 0:ncomp]; yts = y[Its];

        # Use a logistic regression classifier
        logreg = LogisticRegression(multi_class='auto', solver='lbfgs', max_iter=10000)
        logreg.fit(Xtr, ytr)

        yprob = logreg.predict_proba(Xts)
        yprob_max = np.amax(yprob, axis=1)
        yprob_max_ind = (yprob == yprob_max[:, None]).astype(int)
        yprob_1d = np.zeros(yts.shape).astype(int)
        #There is probably a more direct way to do this
        for idx in range(1,yprob_max_ind.shape[1]):
            yprob_1d += (yprob_max_ind[:,idx]*idx).astype(int)
        acc_cf = (yts == yprob_1d).astype(int)
        #print('yts=',yts)
        #print('yprob_1d=',yprob_1d)
        #acc_cf = (yts == ypred).astype(int)
        acc[icomp, ifold] = np.mean(acc_cf)
        print("...kfold",ifold,"ncomp=",ncomp,"Accur=",acc[icomp,ifold])
    #end of inner for()
    acc_comp[icomp] = np.mean(acc[icomp,:])
```

```
    print("***Mean accuracy folding was",acc_comp[icomp],"(nc=",ncomp,")")  
#end of outer for()
```

```

We will try 16 # of PCs: [ 1 2 3 4 5 6 7 8 9 10 20 30 4
0 50 100 150]
X.shape= (200, 150)
:::-----
:::Trying (icomp,ncomp) = ( 0 , 1 )
...kfold 0 ncomp= 1 Acur= 0.475
...kfold 1 ncomp= 1 Acur= 0.5
...kfold 2 ncomp= 1 Acur= 0.375
...kfold 3 ncomp= 1 Acur= 0.5
...kfold 4 ncomp= 1 Acur= 0.475
***Mean accuracy folding was 0.465 (nc= 1 )
:::-----
:::Trying (icomp,ncomp) = ( 1 , 2 )
...kfold 0 ncomp= 2 Acur= 0.9
...kfold 1 ncomp= 2 Acur= 0.925
...kfold 2 ncomp= 2 Acur= 0.95
...kfold 3 ncomp= 2 Acur= 0.925
...kfold 4 ncomp= 2 Acur= 0.9
***Mean accuracy folding was 0.9200000000000002 (nc= 2 )
:::-----
:::Trying (icomp,ncomp) = ( 2 , 3 )
...kfold 0 ncomp= 3 Acur= 0.9
...kfold 1 ncomp= 3 Acur= 0.975
...kfold 2 ncomp= 3 Acur= 0.95
...kfold 3 ncomp= 3 Acur= 0.85
...kfold 4 ncomp= 3 Acur= 0.925
***Mean accuracy folding was 0.9200000000000002 (nc= 3 )
:::-----
:::Trying (icomp,ncomp) = ( 3 , 4 )
...kfold 0 ncomp= 4 Acur= 0.95
...kfold 1 ncomp= 4 Acur= 0.85
...kfold 2 ncomp= 4 Acur= 0.975
...kfold 3 ncomp= 4 Acur= 0.9
...kfold 4 ncomp= 4 Acur= 0.95
***Mean accuracy folding was 0.925 (nc= 4 )
:::-----
:::Trying (icomp,ncomp) = ( 4 , 5 )
...kfold 0 ncomp= 5 Acur= 0.975
...kfold 1 ncomp= 5 Acur= 0.9
...kfold 2 ncomp= 5 Acur= 0.875
...kfold 3 ncomp= 5 Acur= 0.85
...kfold 4 ncomp= 5 Acur= 1.0
***Mean accuracy folding was 0.9199999999999999 (nc= 5 )
:::-----
:::Trying (icomp,ncomp) = ( 5 , 6 )
...kfold 0 ncomp= 6 Acur= 0.875
...kfold 1 ncomp= 6 Acur= 0.95
...kfold 2 ncomp= 6 Acur= 0.95
...kfold 3 ncomp= 6 Acur= 0.95
...kfold 4 ncomp= 6 Acur= 0.95
***Mean accuracy folding was 0.9349999999999999 (nc= 6 )
:::-----
:::Trying (icomp,ncomp) = ( 6 , 7 )
...kfold 0 ncomp= 7 Acur= 0.975
...kfold 1 ncomp= 7 Acur= 0.975
...kfold 2 ncomp= 7 Acur= 0.95
...kfold 3 ncomp= 7 Acur= 0.875

```

```
...kfold 4 ncomp= 7 Accur= 0.875
***Mean accuracy folding was 0.93 (nc= 7 )
:::-----
:::Trying (icomp,ncomp) = ( 7 , 8 )
...kfold 0 ncomp= 8 Accur= 1.0
...kfold 1 ncomp= 8 Accur= 0.975
...kfold 2 ncomp= 8 Accur= 0.925
...kfold 3 ncomp= 8 Accur= 0.875
...kfold 4 ncomp= 8 Accur= 0.875
***Mean accuracy folding was 0.93 (nc= 8 )
:::-----
:::Trying (icomp,ncomp) = ( 8 , 9 )
...kfold 0 ncomp= 9 Accur= 0.975
...kfold 1 ncomp= 9 Accur= 0.95
...kfold 2 ncomp= 9 Accur= 0.975
...kfold 3 ncomp= 9 Accur= 0.875
...kfold 4 ncomp= 9 Accur= 0.9
***Mean accuracy folding was 0.9349999999999999 (nc= 9 )
:::-----
:::Trying (icomp,ncomp) = ( 9 , 10 )
...kfold 0 ncomp= 10 Accur= 0.9
...kfold 1 ncomp= 10 Accur= 0.9
...kfold 2 ncomp= 10 Accur= 0.925
...kfold 3 ncomp= 10 Accur= 0.975
...kfold 4 ncomp= 10 Accur= 0.95
***Mean accuracy folding was 0.93 (nc= 10 )
:::-----
:::Trying (icomp,ncomp) = ( 10 , 20 )
...kfold 0 ncomp= 20 Accur= 0.95
...kfold 1 ncomp= 20 Accur= 0.95
...kfold 2 ncomp= 20 Accur= 0.9
...kfold 3 ncomp= 20 Accur= 0.95
...kfold 4 ncomp= 20 Accur= 0.925
***Mean accuracy folding was 0.9349999999999999 (nc= 20 )
:::-----
:::Trying (icomp,ncomp) = ( 11 , 30 )
...kfold 0 ncomp= 30 Accur= 0.95
...kfold 1 ncomp= 30 Accur= 0.925
...kfold 2 ncomp= 30 Accur= 0.975
...kfold 3 ncomp= 30 Accur= 0.925
...kfold 4 ncomp= 30 Accur= 0.95
***Mean accuracy folding was 0.9450000000000001 (nc= 30 )
:::-----
:::Trying (icomp,ncomp) = ( 12 , 40 )
...kfold 0 ncomp= 40 Accur= 0.95
...kfold 1 ncomp= 40 Accur= 0.925
...kfold 2 ncomp= 40 Accur= 0.925
...kfold 3 ncomp= 40 Accur= 0.975
...kfold 4 ncomp= 40 Accur= 0.95
***Mean accuracy folding was 0.945 (nc= 40 )
:::-----
:::Trying (icomp,ncomp) = ( 13 , 50 )
...kfold 0 ncomp= 50 Accur= 0.925
...kfold 1 ncomp= 50 Accur= 0.975
...kfold 2 ncomp= 50 Accur= 0.9
...kfold 3 ncomp= 50 Accur= 0.975
...kfold 4 ncomp= 50 Accur= 0.95
```

```
***Mean accuracy folding was 0.945 (nc= 50 )
:::-----
:::Trying (icomp,ncomp) = ( 14 , 100 )
...kfold 0 ncomp= 100 Accur= 0.95
...kfold 1 ncomp= 100 Accur= 0.95
...kfold 2 ncomp= 100 Accur= 0.925
...kfold 3 ncomp= 100 Accur= 0.925
...kfold 4 ncomp= 100 Accur= 0.975
***Mean accuracy folding was 0.945 (nc= 100 )
:::-----
:::Trying (icomp,ncomp) = ( 15 , 150 )
...kfold 0 ncomp= 150 Accur= 0.95
...kfold 1 ncomp= 150 Accur= 0.925
...kfold 2 ncomp= 150 Accur= 0.975
...kfold 3 ncomp= 150 Accur= 0.925
...kfold 4 ncomp= 150 Accur= 0.95
***Mean accuracy folding was 0.9450000000000001 (nc= 150 )
```

Results were not bad at all. We got around 95% of accuracy. Below we print the results and choose the number of PCs using the SE RULE and the NORMAL RULE.

Please note that some waveforms may be harder to detect than others. To check that we will not use a Multi-class Logistic Regression later

```
In [13]: acc_mean = np.mean(acc, axis=1)
acc_se    = np.std(acc, axis=1) / np.sqrt(nfold-1)
plt.figure(figsize=(10,8))
plt.errorbar(ncomp_test, acc_mean, yerr=acc_se)

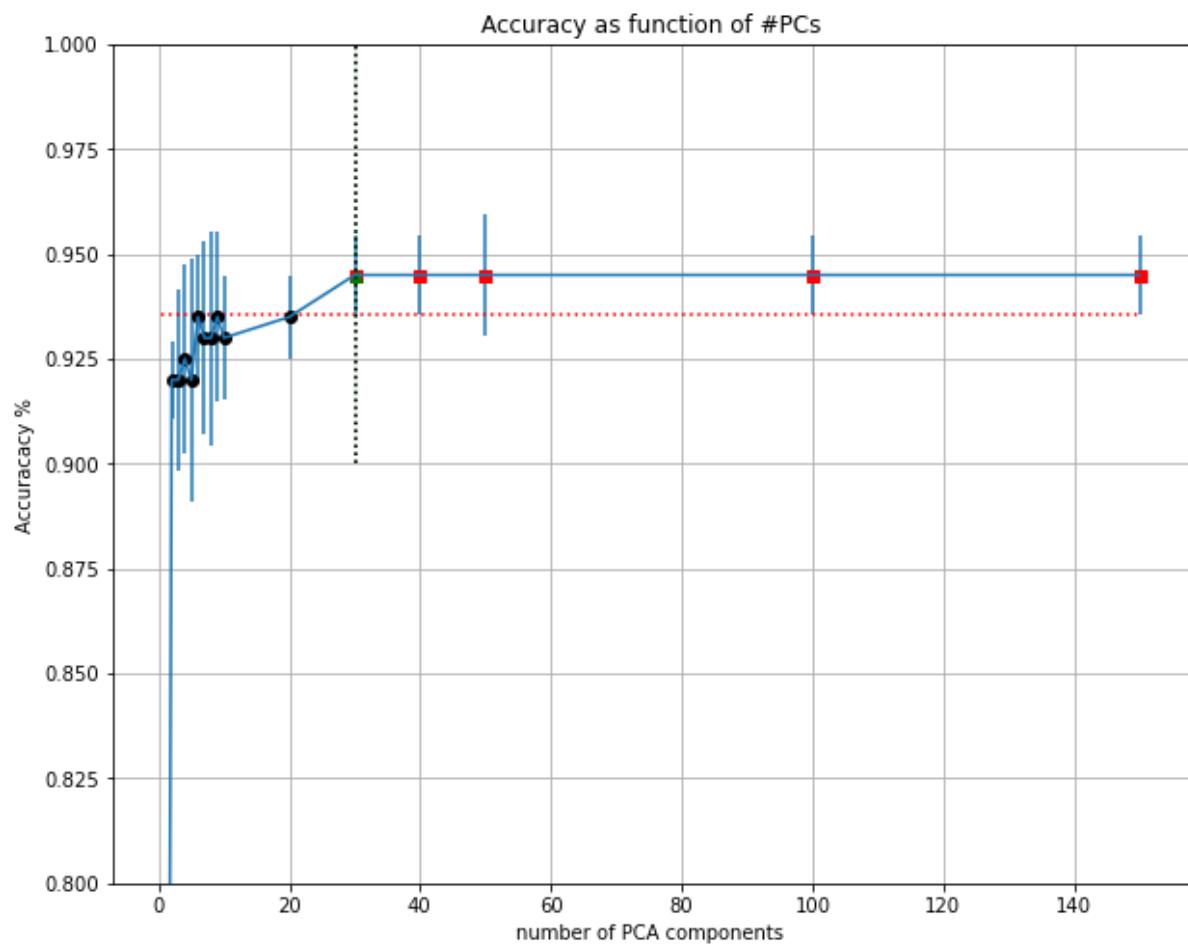
# Optimal order with the normal rule
# Optimal order with one SE rule
# Find the max R^2
imax      = np.argmax(acc_mean)
acc_tgt = acc_mean[imax] - acc_se[imax]

# Find the Lowest delay above the target
I = np.where(acc_mean >= acc_tgt)[0]
iopt = I[0]
dopt = ncomp_test[iopt]
print(imax, iopt, dopt, acc_tgt)
# Plot the line at the target level
plt.plot([acc_mean[0],ncomp_test[len(ncomp_test)-1]],[acc_tgt,acc_tgt], 'r:')
plt.plot(ncomp_test, acc_mean, 'ko')
plt.plot(ncomp_test[I], acc_mean[I], 'rs')
plt.plot(ncomp_test[imax], acc_mean[imax], 'g^')
# Plot the line at the optimal model order
plt.plot([dopt,dopt], [0.9,1], 'g:')
plt.plot([ncomp_test[imax],ncomp_test[imax]], [0.9,1], 'k:')

plt.title('Accuracy as function of #PCs')
plt.xlabel('number of PCA components')
plt.ylabel('Accuracacy %')
plt.ylim((0.8,1))
plt.grid()

# Print results
print("SE      RULE: estimated optimal order #pcs=%d"%dopt)
print("NORMAL RULE: estimated optimal order #pcs=%d"%ncomp_test[imax])
```

```
11 11 30 0.9356458565330652
SE      RULE: estimated optimal order #pcs=30
NORMAL RULE: estimated optimal order #pcs=30
```



NOW we will do a multi-class Logistic Regression.
Instead of checking for the presence of the waveform or not embedded the gaussian noise, we will try to detect whether a specific waveform is present.

We will see that the results are not as accurate as in the binary (waveform present or not) case.

FIRST, we need to generate the proper labels for the data. There will be 6 labels, one for no waveform present and 5 more for the possible signals that may be being transmitted.

```
In [14]: y = subsetRadarStatus.ravel()
WLAB = ['NOISE_ONLY', 'P0N#1', 'P0N#2', 'Q3N#1', 'Q3N#2', 'Q3N#3']
#aux=subsetInfo['BinNo'];print("WAVEFORMS in the file:", aux.unique());
I1=np.where(subsetInfo['BinNo']==WLAB[1])
I2=np.where(subsetInfo['BinNo']==WLAB[2])
I3=np.where(subsetInfo['BinNo']==WLAB[3])
I4=np.where(subsetInfo['BinNo']==WLAB[4])
I5=np.where(subsetInfo['BinNo']==WLAB[5])
y6 = np.zeros(X.shape[0])
y6[I1] = 1
y6[I2] = 2
y6[I3] = 3
y6[I4] = 4
y6[I5] = 5

print('ASCII WAVEFORM LABELS:', WLAB)
print('NUMERIC WAVEFORM LABELS:\n', y6)
print('CHECKING (positions should match with above):\n',subsetInfo['BinNo'])
```

ASCII WAVEFORM LABELS: ['NOISE_ONLY', 'P0N#1', 'P0N#2', 'Q3N#1', 'Q3N#2', 'Q3N#3']
 NUMERIC WAVEFORM LABELS:
 [0. 0. 5. 2. 0.
 0. 0. 3. 0. 0. 4. 1. 0. 0. 0. 0. 2. 0. 2. 3. 1. 2. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 2. 0. 0. 0. 0. 0. 0. 5. 5. 5. 0. 0. 0. 0. 0. 1. 5. 1. 3. 0. 0. 2.
 5. 0. 3. 2. 0. 2. 0. 2. 0. 0. 0. 3. 0. 3. 2. 0. 3. 0. 0. 2. 2. 2. 0. 0.
 0. 0. 5. 0. 0. 4. 5. 4. 1. 4. 1. 3. 2. 2. 0. 4. 0. 0. 1. 1. 0. 4. 5. 1.
 0. 0. 0. 0. 4. 3. 0. 0. 3. 0. 0. 1. 0. 0. 5. 0. 0. 0. 2. 4. 1. 0. 2. 4. 4.
 0. 5. 0. 5. 0. 3. 3. 0. 2. 0. 0. 0. 0. 4. 0. 2. 1. 5. 5. 0. 5. 0. 1. 3.
 0. 0. 4. 0. 4. 4. 1. 0. 0. 5. 5. 2. 2. 0. 0. 4. 2. 0. 5. 0. 3. 0. 0. 0.
 4. 5. 1. 3. 4. 3. 0. 2.]
 CHECKING (positions should match with above):
 0 NaN
 1 NaN
 2 Q3N#3
 3 P0N#2
 4 NaN
 ...
 195 Q3N#1
 196 Q3N#2
 197 Q3N#1
 198 NaN
 199 P0N#2
 Name: BinNo, Length: 200, dtype: object

```
In [15]: #from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold

y = y6
X = MAT_S #The PCs

# Create a K-fold object
nfold = 5
kf = KFold(n_splits=nfold, shuffle=True)
kf.get_n_splits(X)

# Number of PCs to try
ncomp_test = np.array([1,2,3,4,5,6,7,8,9,10,20,30,40,50,100,X.shape[1]])
#ncomp_test = np.arange(2,2 +1)
num_nc = len(ncomp_test)
print("We will try %d # of PCs: "%num_nc, ncomp_test)
print('X.shape=', X.shape)

# Accuracy: acc[icomp,ifold] is test accuracy when
# using `ncomp = ncomp_test[icomp]` in fold `ifold`.
acc = np.zeros((num_nc,nfold))
acc_comp = np.zeros((num_nc))

# Loop over number of components to test
for icomp, ncomp in enumerate(ncomp_test):
    print(":::-----")
    print(":::Trying (icomp,ncomp) = (", icomp, ',', ncomp, ')')
    # Look over the folds
    for ifold, I in enumerate(kf.split(X)):
        Itr, Its = I

        # Xtr, Xts, ytr, yts = ...
        Xtr = X[Itr, 0:ncomp]; ytr = y[Itr];
        Xts = X[Its, 0:ncomp]; yts = y[Its];

        # Use a logistic regression classifier
        logreg = LogisticRegression(multi_class='auto', solver='lbfgs', max_iter=100000)
        logreg.fit(Xtr, ytr)

        yprob = logreg.predict_proba(Xts)
        yprob_max = np.amax(yprob, axis=1)
        yprob_max_ind = (yprob == yprob_max[:, None]).astype(int)
        yprob_1d = np.zeros(yts.shape).astype(int)
        #There is probably a more direct way to do this
        for idx in range(1,yprob_max_ind.shape[1]):
            yprob_1d += (yprob_max_ind[:,idx]*idx).astype(int)
        acc_cf = (yts == yprob_1d).astype(int)
        #print('yts=',yts)
        #print('yprob_1d=',yprob_1d)
        #acc_cf = (yts == ypred).astype(int)
        acc[icomp, ifold] = np.mean(acc_cf)
        print("...kfold",ifold,"ncomp=",ncomp,"Accur=",acc[icomp,ifold])
    #end of inner for()
    acc_comp[icomp] = np.mean(acc[icomp,:])
```

```
    print("***Mean accuracy folding was",acc_comp[icomp],"(nc=",ncomp,")")  
#end of outer for()
```

```
We will try 16 # of PCs: [ 1 2 3 4 5 6 7 8 9 10 20 30 4
0 50 100 150]
X.shape= (200, 150)
:::-----
:::Trying (icomp,ncomp) = ( 0 , 1 )
...kfold 0 ncomp= 1 Accur= 0.45
...kfold 1 ncomp= 1 Accur= 0.625
...kfold 2 ncomp= 1 Accur= 0.575
...kfold 3 ncomp= 1 Accur= 0.45
...kfold 4 ncomp= 1 Accur= 0.475
***Mean accuracy folding was 0.515 (nc= 1 )
:::-----
:::Trying (icomp,ncomp) = ( 1 , 2 )
...kfold 0 ncomp= 2 Accur= 0.7
...kfold 1 ncomp= 2 Accur= 0.55
...kfold 2 ncomp= 2 Accur= 0.65
...kfold 3 ncomp= 2 Accur= 0.725
...kfold 4 ncomp= 2 Accur= 0.55
***Mean accuracy folding was 0.635 (nc= 2 )
:::-----
:::Trying (icomp,ncomp) = ( 2 , 3 )
...kfold 0 ncomp= 3 Accur= 0.65
...kfold 1 ncomp= 3 Accur= 0.8
...kfold 2 ncomp= 3 Accur= 0.55
...kfold 3 ncomp= 3 Accur= 0.625
...kfold 4 ncomp= 3 Accur= 0.725
***Mean accuracy folding was 0.67 (nc= 3 )
:::-----
:::Trying (icomp,ncomp) = ( 3 , 4 )
...kfold 0 ncomp= 4 Accur= 0.7
...kfold 1 ncomp= 4 Accur= 0.775
...kfold 2 ncomp= 4 Accur= 0.6
...kfold 3 ncomp= 4 Accur= 0.675
...kfold 4 ncomp= 4 Accur= 0.6
***Mean accuracy folding was 0.67 (nc= 4 )
:::-----
:::Trying (icomp,ncomp) = ( 4 , 5 )
...kfold 0 ncomp= 5 Accur= 0.725
...kfold 1 ncomp= 5 Accur= 0.725
...kfold 2 ncomp= 5 Accur= 0.625
...kfold 3 ncomp= 5 Accur= 0.65
...kfold 4 ncomp= 5 Accur= 0.65
***Mean accuracy folding was 0.675 (nc= 5 )
:::-----
:::Trying (icomp,ncomp) = ( 5 , 6 )
...kfold 0 ncomp= 6 Accur= 0.65
...kfold 1 ncomp= 6 Accur= 0.675
...kfold 2 ncomp= 6 Accur= 0.725
...kfold 3 ncomp= 6 Accur= 0.575
...kfold 4 ncomp= 6 Accur= 0.7
***Mean accuracy folding was 0.665 (nc= 6 )
:::-----
:::Trying (icomp,ncomp) = ( 6 , 7 )
...kfold 0 ncomp= 7 Accur= 0.575
...kfold 1 ncomp= 7 Accur= 0.65
...kfold 2 ncomp= 7 Accur= 0.675
...kfold 3 ncomp= 7 Accur= 0.75
```

```
...kfold 4 ncomp= 7 Accur= 0.775
***Mean accuracy folding was 0.685 (nc= 7 )
:::-----
:::Trying (icomp,ncomp) = ( 7 , 8 )
...kfold 0 ncomp= 8 Accur= 0.7
...kfold 1 ncomp= 8 Accur= 0.7
...kfold 2 ncomp= 8 Accur= 0.675
...kfold 3 ncomp= 8 Accur= 0.625
...kfold 4 ncomp= 8 Accur= 0.625
***Mean accuracy folding was 0.665 (nc= 8 )
:::-----
:::Trying (icomp,ncomp) = ( 8 , 9 )
...kfold 0 ncomp= 9 Accur= 0.7
...kfold 1 ncomp= 9 Accur= 0.75
...kfold 2 ncomp= 9 Accur= 0.575
...kfold 3 ncomp= 9 Accur= 0.6
...kfold 4 ncomp= 9 Accur= 0.7
***Mean accuracy folding was 0.665 (nc= 9 )
:::-----
:::Trying (icomp,ncomp) = ( 9 , 10 )
...kfold 0 ncomp= 10 Accur= 0.7
...kfold 1 ncomp= 10 Accur= 0.725
...kfold 2 ncomp= 10 Accur= 0.7
...kfold 3 ncomp= 10 Accur= 0.55
...kfold 4 ncomp= 10 Accur= 0.65
***Mean accuracy folding was 0.6649999999999999 (nc= 10 )
:::-----
:::Trying (icomp,ncomp) = ( 10 , 20 )
...kfold 0 ncomp= 20 Accur= 0.75
...kfold 1 ncomp= 20 Accur= 0.65
...kfold 2 ncomp= 20 Accur= 0.575
...kfold 3 ncomp= 20 Accur= 0.675
...kfold 4 ncomp= 20 Accur= 0.65
***Mean accuracy folding was 0.6599999999999999 (nc= 20 )
:::-----
:::Trying (icomp,ncomp) = ( 11 , 30 )
...kfold 0 ncomp= 30 Accur= 0.7
...kfold 1 ncomp= 30 Accur= 0.75
...kfold 2 ncomp= 30 Accur= 0.625
...kfold 3 ncomp= 30 Accur= 0.65
...kfold 4 ncomp= 30 Accur= 0.7
***Mean accuracy folding was 0.6849999999999999 (nc= 30 )
:::-----
:::Trying (icomp,ncomp) = ( 12 , 40 )
...kfold 0 ncomp= 40 Accur= 0.525
...kfold 1 ncomp= 40 Accur= 0.75
...kfold 2 ncomp= 40 Accur= 0.775
...kfold 3 ncomp= 40 Accur= 0.65
...kfold 4 ncomp= 40 Accur= 0.6
***Mean accuracy folding was 0.6599999999999999 (nc= 40 )
:::-----
:::Trying (icomp,ncomp) = ( 13 , 50 )
...kfold 0 ncomp= 50 Accur= 0.675
...kfold 1 ncomp= 50 Accur= 0.725
...kfold 2 ncomp= 50 Accur= 0.625
...kfold 3 ncomp= 50 Accur= 0.65
...kfold 4 ncomp= 50 Accur= 0.7
```

```
***Mean accuracy folding was 0.675 (nc= 50 )
:::-----
:::Trying (icomp,ncomp) = ( 14 , 100 )
...kfold 0 ncomp= 100 Accur= 0.725
...kfold 1 ncomp= 100 Accur= 0.6
...kfold 2 ncomp= 100 Accur= 0.6
...kfold 3 ncomp= 100 Accur= 0.775
...kfold 4 ncomp= 100 Accur= 0.775
***Mean accuracy folding was 0.695 (nc= 100 )
:::-----
:::Trying (icomp,ncomp) = ( 15 , 150 )
...kfold 0 ncomp= 150 Accur= 0.8
...kfold 1 ncomp= 150 Accur= 0.625
...kfold 2 ncomp= 150 Accur= 0.7
...kfold 3 ncomp= 150 Accur= 0.675
...kfold 4 ncomp= 150 Accur= 0.575
***Mean accuracy folding was 0.675 (nc= 150 )
```

As we can see the results were not excellent. However, some classification was still possible.

Below we show the how the number of PCs affects the multi-class logistic regression case according to the SE and NORMAL rules. Because now the standard errors are larger, the number of PCs vary more from run to run.

```
In [16]: acc_mean = np.mean(acc, axis=1)
acc_se    = np.std(acc, axis=1) / np.sqrt(nfold-1)
plt.figure(figsize=(10,8))
plt.errorbar(ncomp_test, acc_mean, yerr=acc_se)

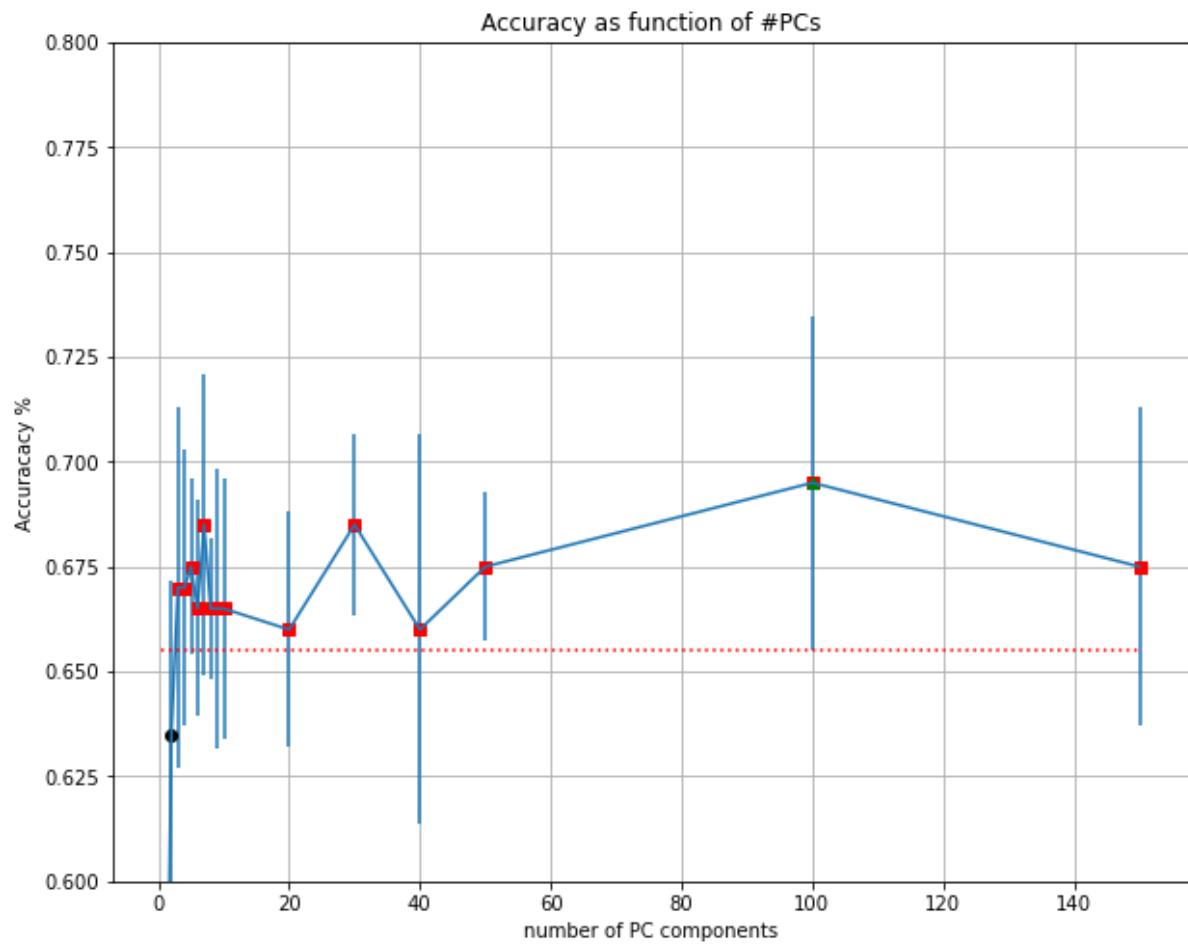
# Optimal order with the normal rule
# Optimal order with one SE rule
# Find the max R^2
imax      = np.argmax(acc_mean)
acc_tgt = acc_mean[imax] - acc_se[imax]

# Find the Lowest delay above the target
I = np.where(acc_mean >= acc_tgt)[0]
iopt = I[0]
dopt = ncomp_test[iopt]
print(imax, iopt, dopt, acc_tgt)
# Plot the line at the target level
plt.plot([acc_mean[0],ncomp_test[len(ncomp_test)-1]],[acc_tgt,acc_tgt], 'r:')
plt.plot(ncomp_test, acc_mean, 'ko')
plt.plot(ncomp_test[I], acc_mean[I], 'rs')
plt.plot(ncomp_test[imax], acc_mean[imax], 'g^')
# Plot the line at the optimal model order
plt.plot([dopt,dopt], [0.9,1], 'g:')
plt.plot([ncomp_test[imax],ncomp_test[imax]], [0.9,1], 'k:')

plt.title('Accuracy as function of #PCs')
plt.xlabel('number of PC components')
plt.ylabel('Accuracacy %')
plt.ylim((0.6,0.8))
plt.grid()

# Print results
print("SE      RULE: estimated optimal order #pcs=%d"%dopt)
print("NORMAL RULE: estimated optimal order #pcs=%d"%ncomp_test[imax])
```

14 2 3 0.6551565563737268
 SE RULE: estimated optimal order #pcs=3
 NORMAL RULE: estimated optimal order #pcs=100



Now we calculate the Confusion Matrix for the multi-class case.

Note that the results were good only for detecting the presence and or not of the signal. There was significant mislabeling of the waveforms when they were present in the signal.

```
In [20]: # ***TODO: Print a confusion matrix
from sklearn.metrics import confusion_matrix

nn = X.shape[0]//2

Xtr = X[:nn, ]; ytr = y[:nn];
Xts = X[nn:, ]; yts = y[nn:];
logreg = LogisticRegression(multi_class='auto', solver='lbfgs', max_iter=10000)
logreg.fit(Xtr, ytr)

yprob      = logreg.predict_proba(Xts)
yprob_max   = np.amax(yprob, axis=1)
yprob_max_ind = (yprob == yprob_max[:, None]).astype(int)
yprob_1d    = np.zeros(yts.shape).astype(int)
#There is probably a more direct way to do this
for idx in range(1,yprob_max_ind.shape[1]):
    yprob_1d += (yprob_max_ind[:,idx]*idx).astype(int)

acc = (yts == yprob_1d).astype(int)
print('Accuracy=',np.mean(acc))

C0 = confusion_matrix(yts,yprob_1d)
# Normalize the confusion matrix
Csum = np.sum(C0, axis=1)
C = C0 / Csum[:, None]
# Print the confusion matrix
print("\nConfusion Matrix - Normalized")
print(np.array_str(C, precision=3, suppress_small=True))
print("\nConfusion Matrix - Not Normalized")
print(np.array_str(C0, precision=3, suppress_small=True))
```

Accuracy= 0.58

Confusion Matrix - Normalized

| | | | | | |
|--------|-------|-------|-------|----|---------|
| [[1. | 0. | 0. | 0. | 0. | 0.]] |
| [0.273 | 0.182 | 0.273 | 0.091 | 0. | 0.182] |
| [0. | 0. | 0.9 | 0.1 | 0. | 0.]] |
| [0.111 | 0.222 | 0.333 | 0.222 | 0. | 0.111] |
| [0.125 | 0.375 | 0.312 | 0.125 | 0. | 0.062] |
| [0.167 | 0.083 | 0.083 | 0.417 | 0. | 0.25]] |

Confusion Matrix - Not Normalized

| | | | | | |
|------|---|---|---|---|-----|
| [[42 | 0 | 0 | 0 | 0 | 0] |
| [3 | 2 | 3 | 1 | 0 | 2] |
| [0 | 0 | 9 | 1 | 0 | 0] |
| [1 | 2 | 3 | 2 | 0 | 1] |
| [2 | 6 | 5 | 2 | 0 | 1] |
| [2 | 1 | 1 | 5 | 0 | 3]] |

For Discussion (Future Work)

We can handle the spectrograms as images and train a Deep Network to classify the signals. This may bring better results than PCA.

One idea is to use transfer learning and reuse a trained network like VGG16 to do the job, only training the final fully connected layer and perhaps some of the initial convolution layers (because the images a bit different). This would be an interesting extension of this project.